D.4.1 Initial report on mechanisms that enable green-oriented zero touch management of CECC resources - Initial

Document Summary Information

Project Identifier	HORIZON-CL4-2022-DATA-01. Project 101093129		
Project name	Agile and Cognitive Cloud-edge Continuum management		
Acronym	AC^3		
Start Date	January 1, 2023	**End Date**	December 31, 2025
Project URL	www.ac3-project.eu		
Deliverable	D4.1. Initial report on mechanisms that enable green-oriented zero touch management of CECC resources - Initial		
Work Package	WP4		
Contractual due date	30th June 2024	**Actual submission date**	30th June 2024
Type	R – Document, Report	**Dissemination Level**	PU – Public
Lead Beneficiary	IBM		
Responsible Author	Amadou Ba (IBM)		
Contributors	Amadou Ba (IBM), Diego Tsutsumi (IBM), Adlen Ksentini (EUR), Ayoub Mokhtari (EUR), Mohamed Mekki (EUR), Sofiane Messaoudi (EUR), Kostas Ramantas (IQU), Ioannis		

Funded by the European Union

AC^3 project has received funding from European Union's Horizon Europe research and innovation programme under Grant Agreement No 101093129.

	Zenginis (IQU), Ben Capper (RHT), Ray Carroll (RHT), Ryan Jenkins (RHT), Christos Verikoukis (ISI/ATH), Elias Dritsas (ISI/ATH), Vasilis Avgerinos (ISI/ATH), Athanasios Kordelas (CTX), George Tsolis (CTX), John Beredimas (CTX), Alabi Rasheed (FIN), Ibrahim Afolabi (FIN), Vrettos Moulos (UPR), Dimitrios Amaxilatis (SPA), Nikos Tsironis (SPA), Ali Nikoukar (ION), Maria Tsoli (UPR), Giannis Koulopoulos (UPR), Giannis Mpekos (UPR), Podimata Anna (UPR)
Peer reviewer(s)	Ibrahim Afolabi (FIN), Nikos Tsironis (SPA)

Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete	Changes	Contributor(s)
V1.0	04/03/2024	5	Initial structure (ToC)	All partners
V2.0	15/04/2024	50	Initial contributions	Section 4 (EUR, CTX, UPR, SPA, RHT) Section 5.1 (FIN) Section 5.2 (SPA) Section 5.3.1 (IBM) Section 5.3.2 (EUR) Section 5.3.3 (IBM) Section 5.3.4 (ATH/ISI) Section 6 (IQU) Section 7 (RHT)
V3.0	07/05/2024	90	Final contributions	Section 1 (IBM) Section 2 (IBM) Section 3 (IBM) Section 4 (EUR, CTX, UPR, SPA, RHT) Section 5.1 (FIN) Section 5.2 (SPA) Section 5.3.1 (IBM) Section 5.3.2 (EUR) Section 5.3.3 (IBM) Section 5.3.4 (ATH/ISI) Section 6 (IQU) Section 7 (RHT) Section 8 (IBM)
V4.0	16/05/2024	95	Internal review	Ibrahim Afolabi (FIN), Nikos Tsironis (SPA)
V4.1	30/05/2024	98	Technical Manager review	Adlen Ksentini (EUR)
V5.0	15/06/2024	100	Project Coordinator review	Christos Verikoukis (ATH/ISI)

Disclaimer

Copyright message

Table of Contents

List of Figures

List of Tables

Glossary of terms and abbreviations used

Abbreviation/Term	Description
AC³	Agile and Cognitive Cloud-edge Continuum
ADC	Application Delivery Controller
AI	Artificial Intelligence
API	Application Programming Interface
ADM	Application Delivery Management
CECC	Cloud Edge Computing Continuum
CECCM	Cloud Edge Computing Continuum Manager
CNI	Container Network Interface
CPE	Customer Platform Engineering
CPX	Container Proxy
COE	Customer Observability Exporter
CRD	Customer Resource Descriptor
DL	Deep Learning
DRL	Deep Reinforcement Learning
GNN	Graph Neural Network
ISP	Internet Service Providers
LCM	Life Cycle Management
LIME	Local Interpretable Model-agnostic Explanations
LMS	Local Management System
LSTM	Long Short-Term Memory
ML	Machine Learning
NBI	Northbound Interface
OCM	Open Cluster Management
PaaS	Platform as a Service
QoS	Quality of Service
QoE	Quality of Experience
RL	Reinforcement Learning

SBI	Southbound Interface
SDN	Software Defined Network
SD-WAN	Software Defined Wide Area Network
SHAP	SHapley Additive exPlanations
SLA	Service Level Agreements
SLO	Service Level Objectives
SNMP	Simple Network Management Protocol
SR	Segment Routing
TFT	Temporal Fusion Transformer
TLS	Transport Layer Security
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
VNF	Virtual Network Function
VRF	Virtual Routing and Forwarding
XAI	Explainable AI

1 Executive Summary

The transition from traditional IT to the CECC concept requires the creation of frameworks and resource management capabilities capable of efficiently managing and continuously optimizing resources. This serves as the core impetus behind the AC^3 project, which aims to introduce a pioneering CECC management framework enabling scalability, agility, and efficiency in resource management. At the heart of the AC^3 project lies the CECCM, a fundamental component facilitating the advancement of application LCM and configuration through self-management and self-configuration, all while satisfying the SLA. To realize this objective, AC^3 considers recent breakthroughs in AI and ML, particularly eXplainable AI (XAI), to deliver an efficient LCM system for CECC resources, ensuring minimal latency and SLA compliance. WP4 is the key pillar of AC^3. It is the place where AI/ML models for resource management are developed (T4.2), and where LCM decisions for resource management are enforced (T4.3), both enabled by the data emanating from the resource discovery and monitoring of the federated infrastructures (T4.1). Additionally, Network Programmability components are developed in this WP4 to ensure dynamic and flexible traffic management for microservices in the CECC infrastructure. This is achieved while maintaining application SLAs through dynamic updates of routes and resources (T4.4). This deliverable outlines our initial efforts regarding WP4, focusing on resource management across CECC. To set the context, we provide an overview of the monitoring system, highlighting the importance of collecting relevant metrics to build the AI-based LCM. The primary areas of innovation for the proposed mechanisms begin with a unified monitoring data model, emphasizing the need for a comprehensive schema that captures dynamic resource states (CPU, memory, storage), classifies resources by type (edge, cloud), tracks energy sources, and determines geographical locations. Then, we architect a mechanism for collecting the monitored data. We explain the data collection processes for both computing and networking. Specifically, we provide data including various performance, health, and usage metrics at the network and application levels. We list the operational and application-specific metrics required for the AI-based LCM. Furthermore, we describe the resource discovery mechanism, focusing on the interaction between resource exposure and discovery (T4.1).

After data collection, we develop efficient approaches for resource management (T4.2). The first combines XGBoost with SHAP for explainable detection of SLA non-compliance. The second performs spatio-temporal prediction of infrastructure usage with the combination of a Graph Neural Network and a sequence-to-sequence learner. The third is an explainable prediction approach that predicts latency and identifies the influence of various features on the prediction. It leverages the Temporal Fusion Transformer (TFT). The TFT is an attention-based architecture providing multi-horizon forecasting and interpretability. The advantage of TFT lies in its multi-horizon prediction, providing users with access to estimates across the entire path, allowing them to optimize their actions at multiple steps in the future (T4.2). For example, after latency prediction, users can assess SLA compliance and identify potential responsible metrics like CPU or memory or both, in case of non-compliance. This facilitates implementing corrective measures, such as increasing CPU, memory, or both (T4.3). After infrastructure availability prediction, a mechanism for runtime management of microservices, such as resource autoscaling and/or duplication of micro-services has been developed (T4.3), we use an RL approach based on a Soft Actor-Critic (SAC) agent. This will allow for proactive orchestration of microservices based on the prediction results. Furthermore, we report our initial work on dynamic and flexible traffic management through Network Programmability (T4.4). With regard to Network Programmability, we focus on Software Defined Networking (SDN) and Container-Based Networking (CBN) (Kubernetes), where we provide the possibility for proactive and reactive network configuration.

Major steps have been taken to develop and implement the fundamental building blocks required in WP4. These steps also lay the foundation for the remaining work in WP4 where the energy aspect will be integrated for the consolidation of both resource prediction and management mechanisms and realize the ultimate objective of green-oriented zero touch management of CECC resources.

2 Introduction

2.1 Scope

This document represents a public deliverable from AC³'s WP4, offering insights into our ongoing progress, notable achievements, and the innovative AI-driven algorithms developed for resource management, with a strong focus on explainability. This deliverable reviews state-of-the-art monitoring systems, AI methodologies applied to infrastructure availability prediction and management, and advancements in Network Programmability.

2.2 Target Audience

This deliverable is tailored for stakeholders involved in AI for resource management across hybrid, multi-cloud, and cloud-edge environments. It details the implementation of innovative AI techniques for predicting infrastructure availability and managing resource usage effectively.

2.3 Mapping AC³ Outputs

The purpose of this section is to map AC³ Grant Agreement commitments, both within the formal Deliverable and Task description, against the project's respective outputs and the work performed.

Table 1: Adherence to AC³ GA Deliverable & Tasks Descriptions

AC³ GA Component Title	AC³ GA Component Outline	Respective Document Chapter(s)	Justification
DELIVERABLE			
D4.1. Initial report on mechanisms that enable green-oriented zero touch management of CECC resources			
TASKS			
Task 4.1 Resource discovery and monitoring of the federated resources	Define the necessary components and mechanisms to monitor and discover the federated CECC infrastructure resources, including the far edge. Determine the unified data model that needs to be exchanged between the different infrastructure providers. Two models will be considered, one for computing and one for networking.	Section 4	The main building blocks present in the GA and pertaining to Task 4.1 have been addressed in Section 4.

	The data model for computing will be extended to include the available computing resources and information on the energy model used by the DC (green or brown) and the best periods suited for energy. Defines a unified discovery resource communication model, allowing the CECCM to have a complete picture of available resources on the federated CEC. Define scalable data collection and monitoring for the federated resources that are in use by the CECCM.		
Task 4.2 AI/ML models for resource management	Develop ML models in order to predict infrastructure usage, energy usage, and resource availability, including the far edge. ML algorithms will focusing mainly on using explainable models. State-of-the-art explainable models will be tested and used.	Section 5	The main building blocks present in the GA and pertaining to Task 4.2 have been addressed in Section 5. The remaining work such as the consideration of the energy aspect will be addressed in the deliverable D4.2.
Task 4.3 Green-oriented LCM decisions for resource management	Devise the necessary algorithms that take as inputs the resource prediction models to guarantee the application's SLA.	Section 6	The main building blocks present in the GA and pertaining to Task 4.3 have been addressed in Section 6. The remaining work will be addressed in the deliverable D4.2.

	Devise the necessary algorithms that take as inputs the resource prediction to optimize resource usage for the federated CECC as well as the energy consumption. The envisioned solutions could be based on multi-objective optimization theory or ML- based decision-making solutions, such as Reinforcement Learning (RL). Devise algorithms that decide when to migrate microservice in order to optimize energy consumption. Build a knowledge mechanism in order to use the explainable models to evaluate the impacts of the considered decisions and derive their efficiency.		
Task 4.4 Networking programmability of CECC	Embrace two technologies SD-WAN and CFN, for WAN and edge, respectively. Define the necessary mechanisms to allow the programmability of both networks by selecting or extending existing SDN controllers. Update the CECCM northbound API to allow the application developer to request an update of the network connectivity. Improve the CECCM to allow for conflict-	Section 4, Section 7	The main building blocks present in the GA and pertaining to Task 4.4 have been addressed in Section 4, Section 7.

	resolution solution that solves contradictory requests or requests that may impact other deployed applications.		

2.4 Deliverable Overview and Report Structure

In this section, we provide the description of D4.1 structure, outlining the respective sections and their content.

Section 3 presents the AC^3 architecture and WP4 components with special focus on the four tasks.

Section 4 presents the resource discovery and monitoring mechanism.

Section 5 focuses on the AI/ML models for infrastructure usage prediction. Special consideration has been given to XAI.

Section 6 provides the decision enforcement mechanism with particular emphasis on a reinforcement learning approach characterized by the Soft Actor-Critic for resource management.

Section 7 is dedicated to Network Programmability.

Section 8 concludes the deliverable D4.1 and introduces the next steps.

3 AC³ Architecture and WP4 Components

Figure 1 presents the high-level architecture of AC³, composed of three planes and characterized by the User Plane, the Management Plane, and the Infrastructure Plane.

- User Plane: it includes the Service Catalogue, the KPI Collection and Exposure mechanism, and the Ontology along with the Semantic-aware Reasoner.
- Management Plane: it represents the fundamental aspect of the CECCM, which deals with the Data Management, the Applications and Resources Management, along with the Adaptation and Federation layer.
- Infrastructure Plane: it is composed of cloud, edge and network resources.

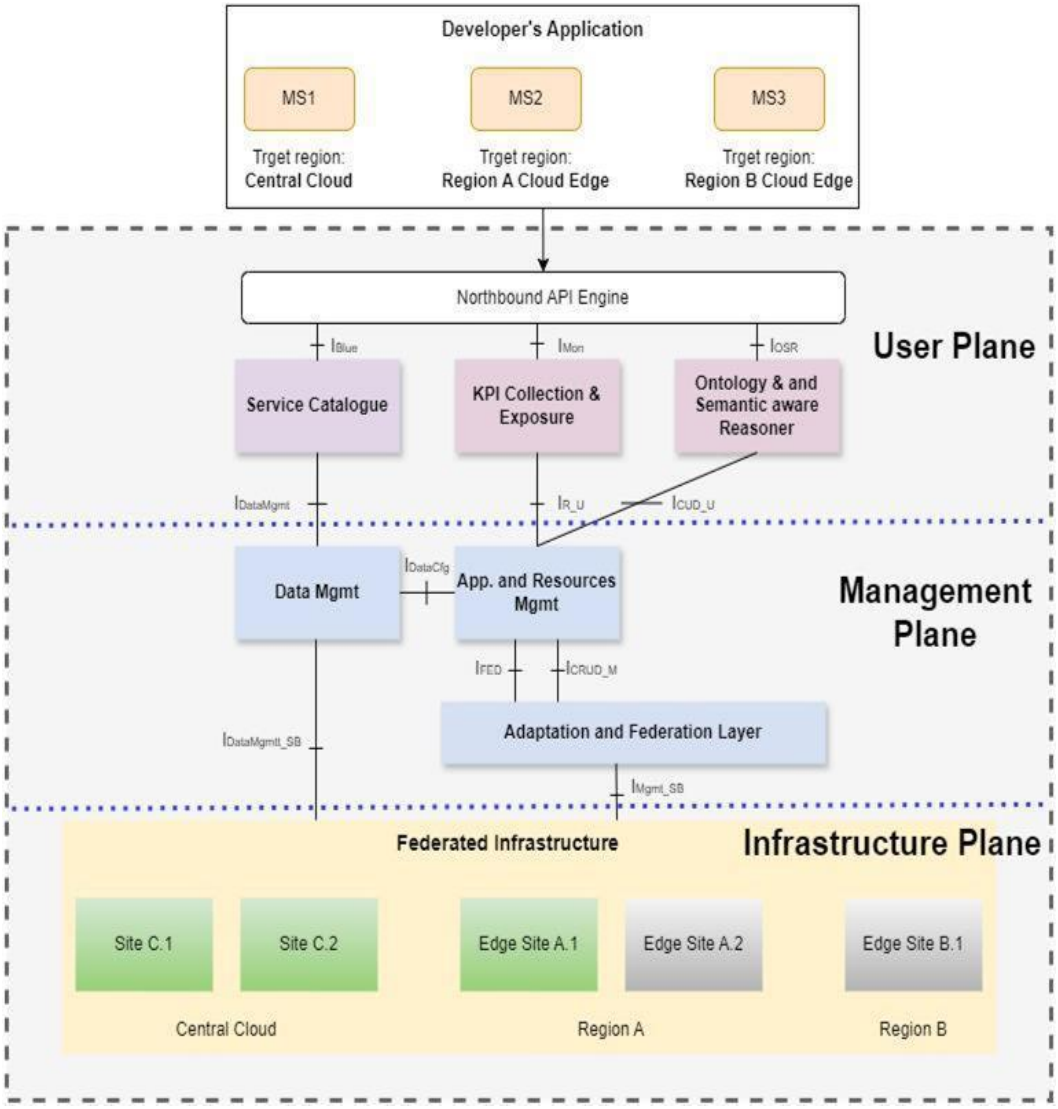

Figure 1: High level architecture of AC³

Figure 2 illustrates the Management Plane showcasing the primary building blocks of WP4. These include the Resource Exposure, Discovery, and Broker alongside the Adaptation Gateway (T4.1), the AI-based CEC Resource Profile (T4.2), the Decision Enforcement (T4.3), and the Network (T4.4). In this deliverable, we delve into the inner workings of these blocks and present the approaches developed for their operationalization.

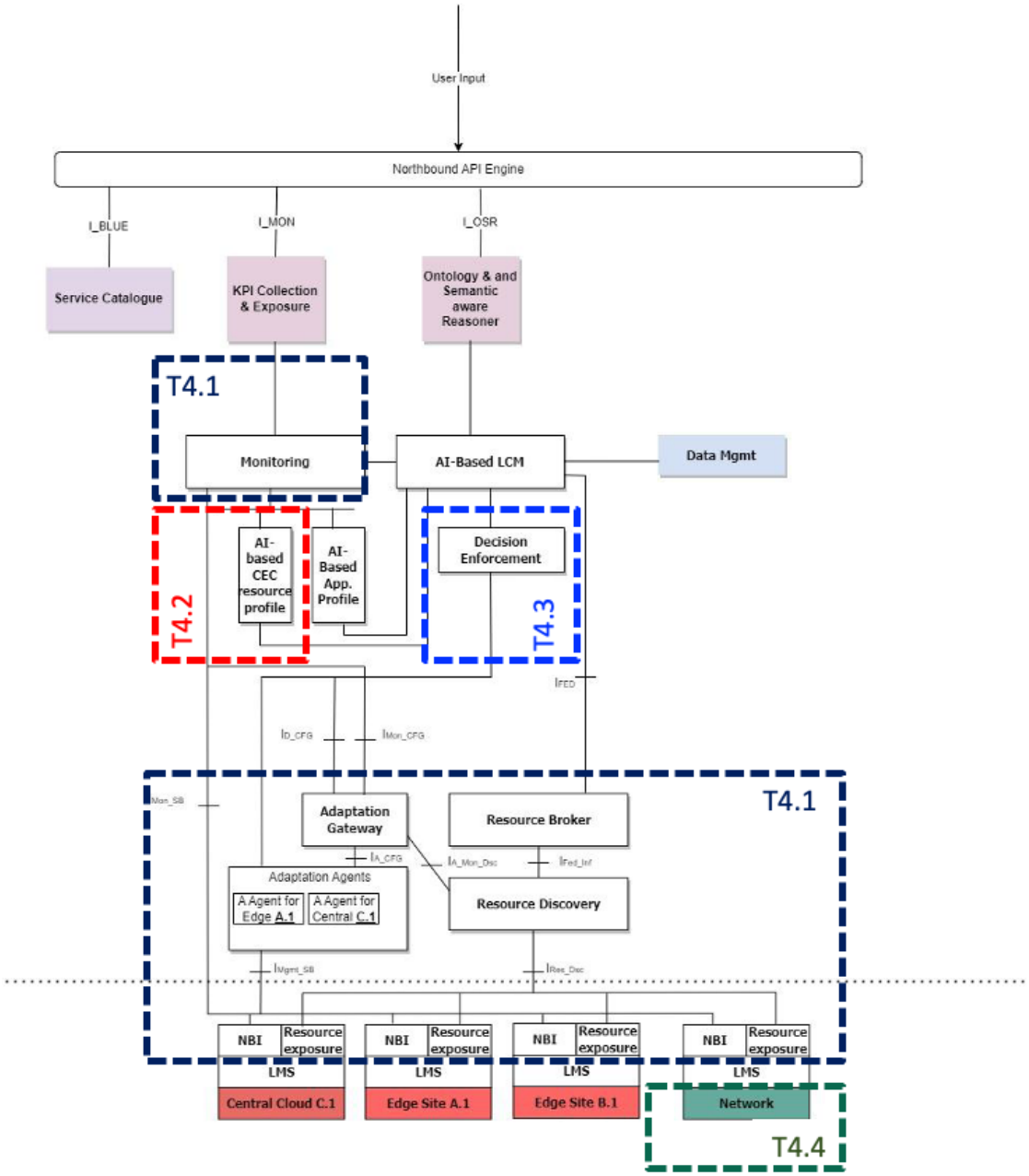

Figure 2: The management plane of AC³

In T4.1 we develop the resource discovering and monitoring approach. T4.1 allows to get the data that will be used in the subsequent tasks. T4.2 is where the XAI approaches for infrastructure usage prediction will be developed. T4.3 addresses resource management using a reinforcement learning approach. T4.4 performs Network Programmability.

4 Resource Discovery and Monitoring

We provide in this section an overview of the monitoring system, a unified monitoring data model, and the mechanism for collecting the monitored data. We explain the data collection processes for both computing and networking.

4.1 Overview of the Monitoring System for the CECCM

Monitoring is a critical component of the CECCM, particularly the AI-based LCM. Indeed, several components of the AI-based LCM rely on the monitoring information collected on the infrastructures as well as on the applications. First, the AI-based application profile uses the collected information on the applications, such as computing resources (CPU, memory, storage) consumption, energy consumed, traffic handled during a period, etc. Using this information, it is possible to build application profiles that will be employed by the AI-based LCM for application management and life-cycle optimization, such as scale down or up resources, or do migration. Second, the AI-based CEC resource profile uses the monitoring information to update the profile of CEC resource used by the CECCM. As CECCM uses resources from the federation, a profile describing in real-time the CEC resource status and profile is vital. The AI-based CEC resource profile will build on the monitoring information using the status of the CEC resource, that is, available computing resources (CPU, memory, storage), type of resource (edge, cloud, networks, far edge), type of energy used (green or brown), the covered locations, etc. All this information will also be used to predict the evolution of CEC resource usage. The AI-based LCM will also use the CEC resources for LCM decisions, such as the initial placement of applications, migration of applications for energy optimization, etc. Both profiles are combined for better resource optimization and SLA guarantee for applications. It should be noted that the monitoring components are activated once a CEC resource from the federation is selected by the CECCM (via the broker). The monitoring component is configured to consume the NBI as exposed by the LMS of the CEC resource.

4.2 Unified Monitoring Data Model and Monitoring Data Collection

4.2.1 Monitoring Data

To effectively build the resource profile that leverages monitoring information to detail the status and characteristics of computing resources, a comprehensive schema that integrates all relevant attributes should be considered. This schema should not only capture the dynamic state of the resources such as CPU, memory, and storage but also classify the resources by type (e.g., edge, cloud), track the source of energy (green or brown), and pinpoint geographical locations.

4.2.2 Enhanced Resource Profile Schema

4.2.2.1 Schema Overview

The schema is designed to provide a detailed snapshot of resource utilization and characteristics across different infrastructure types, energy sources, and locations. This will help in optimizing resource allocation, ensuring sustainability, and improving deployment strategies.

```
{
  "resourceId": "unique_identifier",
  "resourceType": {
    "computing": "VM/Container/Server",
    "infrastructure": "Edge/Cloud/Far Edge",
    "network": "5G/Wi-Fi/Fiber"
  },
  "location": {
```

```
  "latitude": "decimal_degrees",
  "longitude": "decimal_degrees",
  "region": "string"
 },
 "energy": {
  "type": "Green/Brown",
  "consumption": "watts"
 },
 "computingResources": {
  "cpu": {
   "totalCores": "number",
   "usedCores": "number",
   "usagePercentage": "percentage"
  },
  "memory": {
   "total": "bytes",
   "used": "bytes",
   "free": "bytes"
  },
  "storage": {
   "total": "bytes",
   "used": "bytes",
   "free": "bytes"
  }
 },
 "network": {
  "bandwidth": {
   "uplink": "Mbps",
   "downlink": "Mbps"
  },
  "latency": "milliseconds"
 },
 "timestamp": "ISO 8601 format"
}
```

The aforementioned schema represents an initial scenario, focusing on fundamental characteristics for monitoring implementation and resources. However, this schema is designed to be extensible, allowing for the inclusion of a wide range of parameters. These additional parameters can enhance the capabilities of AI-based LCM used for more precise matchmaking. By enriching the profile with detailed attributes, the LCM can more accurately identify and select the best-suited resources from within the federation.

The parameters are organized into categories to enhance manageability and improve readability. Additionally, applications can retrieve only the relevant portion of the JSON file needed for a specific task. This approach not only reduces parsing time but also simplifies the process of querying parameters.

The list of the parameters are:

Resource Identification:

- **resourceId:** A unique identifier for each resource to track different monitoring points.
- **resourceType:** Classifies the resource into categories such as computing (e.g., VM, container), infrastructure type (e.g., edge, cloud), and network type.

Location:

- **Geographical data:** Essential for distributed systems, particularly in edge computing, to optimize data routing and compliance with local regulations.

Energy:

- **Type of energy:** This is crucial for sustainability reports and operational optimizations, distinguishing between green (renewable) and brown (non-renewable) energy sources.
- **Consumption:** Measures how much energy each resource uses, important for cost and environmental impact assessments.

Computing Resources:

- Details the allocation and usage of CPU, memory, and storage, which are critical for performance monitoring and capacity planning.

Network Characteristics:

- **Bandwidth and latency:** These metrics are vital for evaluating network health and performance, particularly in environments where real-time data processing is crucial. Another aspect of it is the necessity in some orchestrators for minimum latency (for example Kubernetes has such a limitation).

Timestamp:

- For time series analysis of resource utilization and tracking changes over time.

On top of that, data will also be monitored by the NetScaler instance. The NetScaler instance refers to a specific deployment of NetScaler, which is a networking product. These data include various metrics related to performance, health, and usage on network and application level. To gather and analyze NetScaler analytics information, the ADM, SNMP monitoring solutions, or custom scripts that query NetScaler APIs can be used. The monitored metrics include:

- **Application Performance:** Monitoring application-specific metrics such as database query times, cache hit rates, response time, error rate, and transaction continuity can help ensure that applications hosted outside of the NetScaler appliance are performing as expected.
- **Response Time:** This is the time taken from NetScaler to respond to a client request. It includes the time spent processing the request internally and communicating with backend servers. Monitoring response time ensures that programs are attentive to person requests.
- **Transaction Throughput:** Transaction throughput measures the charge at which the NetScaler equipment procedures customer requests and forwards the information to backend servers. Monitoring transaction throughput allows discovering bottlenecks and guaranteeing that the NetScaler can handle the current workload efficiently.
- **Error Rates:** tracks the frequency of errors occurring within the application infrastructure, such as connection failures, HTTP errors, SSL errors, etc. Common mistakes include HTTP errors (e.g., 404 Not Found) and application-particular errors. High error rates may suggest troubles with the application or backend servers.
- **SSL connections**: For HTTPS traffic, monitoring SSL connections can provide insights into SSL handsake operations, certificate expiration, and SSL cipher usage.
- **SSL Handshake Time**: ensure that SSL/TLS connections are installed quickly and efficiently.
- **Connection Persistence**: monitoring connection stability guarantees that customer requests are consistently routed to the same backend server across more than one requests. This is vital for programs that require session endurance like shopping carts in e-commerce websites.
- Additional generic metrics that can be used include:
- **Throughput:** measures the amount of data passing through NetScaler at any time.

- **Network latency:** measures the time it takes for a data packet to travel from the client to the server and back. High latency can negatively affect the user experience, so monitoring and optimization is important.
- **Connection Stats:** monitoring metrics like active connections, connection failures, connection retries, and connection timeouts can help identify potential communication issues.
- **Security metrics:** monitoring security-related metrics such as DDoS attack attempts, intrusion attempts, and VPN usage can help identify and mitigate security threats.
- **Load Balancer Metrics:** When the NetScaler appliance is used as a load balancer, monitoring metrics such as server health, request delivery, and server response time can help improve the performance of the load balancer.

4.2.3 Metrics for Monitoring Data Processing Components

The Data Processing components within the AC^3 Data Management Platform as a Service (PaaS) are engineered to supply a dual set of metrics that are instrumental to the AC^3 Monitoring Infrastructure. These metrics are pivotal for maintaining and enhancing the system's performance and reliability. By meticulously monitoring these metrics, AC^3 can ensure that its Data Management PaaS remains robust, efficient, and responsive to the evolving needs of its users, thereby maintaining a high standard of service quality.

4.2.3.1 Operational Metrics

The initial category of metrics focuses on the operational aspects of data processing, specifically addressing the volume, velocity, and latency. Several specific metrics are gathered from the Data Processing components to provide a comprehensive view of the system's performance:

- **Volume:** This metric measures the quantity of data processed by each component within a given timeframe. It helps in understanding the scale of data handling and is crucial for capacity planning and resource allocation.
    - **Data Volume In:** It measures the total volume of incoming data, providing an overview of the workload handled by the system.
    - **Data Volume Out:** Similar to **Data Volume In**, this metric measures the total volume of data outputted by the system, which is crucial for understanding the system's throughput.
- **Velocity:** This metric gauges the speed at which data flows through the components, indicating how swiftly the system can process and transmit data. It's vital for evaluating the system's capability to handle real-time data processing and for optimizing throughput.
    - **Data Points In:** This metric tracks the number of individual data points entering the system, offering insights into the data influx rate.
    - **Data Points Out:** This indicates the number of data points successfully processed and sent out by the system, reflecting the output efficiency.
- **Latency:** This measures the time delay in data processing within each component. Minimizing latency is essential for time-sensitive applications, ensuring that data processing and delivery are accomplished promptly.
- **Resource Usage:** This metric encompasses the consumption of computational resources by each data processing component within the system. It's divided into several key sub-metrics.
    - **CPU/Memory Usage for Broker, Mapper and Manipulator Add-ons:** Monitoring the CPU and memory usage of specific components like Mapper and Broker add-ons is essential for resource optimization and ensuring the system's stability and performance.

4.2.3.2 Application Metrics

The second set of metrics is tailored for application developers. These metrics enable developers to monitor and

optimize the performance of their applications and also provide valuable feedback to AC^3 on its operational performance:

- **Application Performance Metrics:** Developers can track specific performance indicators related to their applications, such as response times, error rates, and transaction volumes. These metrics are crucial for diagnosing and improving application behavior.
- **Feedback Metrics:** By providing feedback on the platform's performance, developers can aid AC^3 in identifying potential improvements or enhancements, ensuring that the platform evolves to meet the users' needs effectively.

4.2.4 Data Collection

The principal mechanism for data collection is designed to be compatible with the Prometheus monitoring solution, a widely recognized tool for gathering and processing operational metrics. For Java-based components, Micrometer[1] is utilized, offering a rich set of functionalities for monitoring application metrics. For Python-based components, werkzeug[2] is employed, a toolkit that facilitates the creation of web applications and includes capabilities for monitoring and data collection. In addtition to the metrics collected by the data management module, Section 4.2.2.1 provides a list of network metrics accompanied by descriptive details, which will also be included in the data collection process.

4.2.5 Data Collection Interfaces for Networking

The Container Proxy (CPX) connects with Prometheus via the Customer Observability Exporter (COE) [1] or directly [2], as shown in Figure 3 and Figure 4. Grafana can be used in both cases to visualize the NetScaler metrics exported to Prometheus for easier interpretation and understanding. The diagram in Figure 3 shows a Prometheus and Grafana integration with NetScaler. COE collects CPX stats like the total hits to a server, HTTP request rate, ssl encryption-decryption rate, etc. from the the Application Delivery Controller (ADC) instances and holds them until the Prometheus server pulls the stats and stores them with a timestamp.

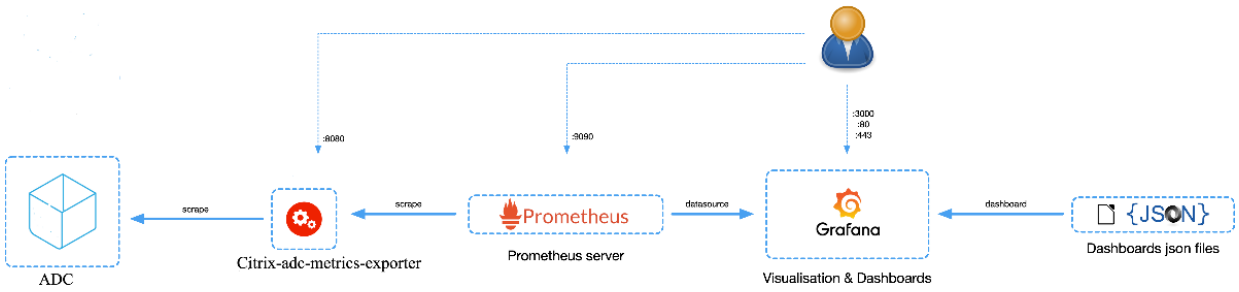

Figure 3: ADC metrics exporter or observability exporter (COE)

As already mentioned, NetScaler now supports directly exporting metrics to Prometheus. For example, CPU and memory usage metrics can be collected to know the NetScaler health. Similarly, metrics like the number of HTTP requests received per second or the number of active clients to monitor application health can be collected. NetScaler supports both the Prometheus pull and push mode. In pull mode, an administrator needs to configure a time series profile that Prometheus queries at regular intervals and pulls the metrics data directly without an exporter resource in between. With pull mode, read-only access can be enabled for a user without superuser privileges to export metrics to Prometheus.

[1] https://www.baeldung.com/micrometer
[2] https://werkzeug.palletsprojects.com/en/3.0.x/

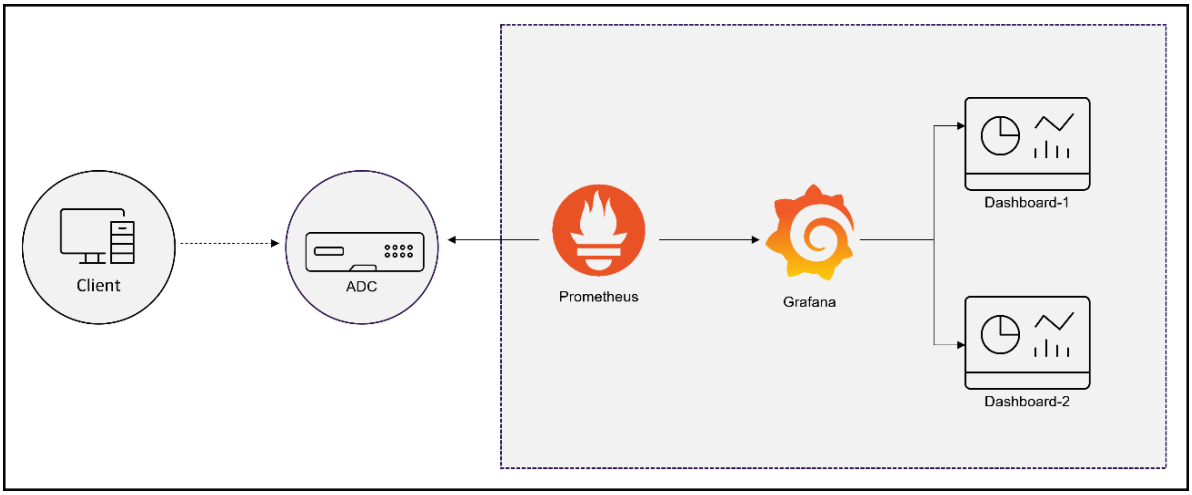

Figure 4: ADC metrics exporter or observability exporter (COE)

4.3 Network Monitoring

Skupper and Submariner can expose numerous types of network monitoring data related to their respective functionalities. Skupper provides monitoring capabilities tailored for inter-cluster connectivity with a Kubernetes environment. Administrators can leverage Prometheus with Skupper to monitor traffic metrics such as latency and packet loss within clusters. Integration with Prometheus allows for easy data collection and visualization of monitoring data, it gathers metrics from endpoints exposed by Skupper components, including routers and proxies. This allows administrators to gain insights into network performance and troubleshoot issues [3].

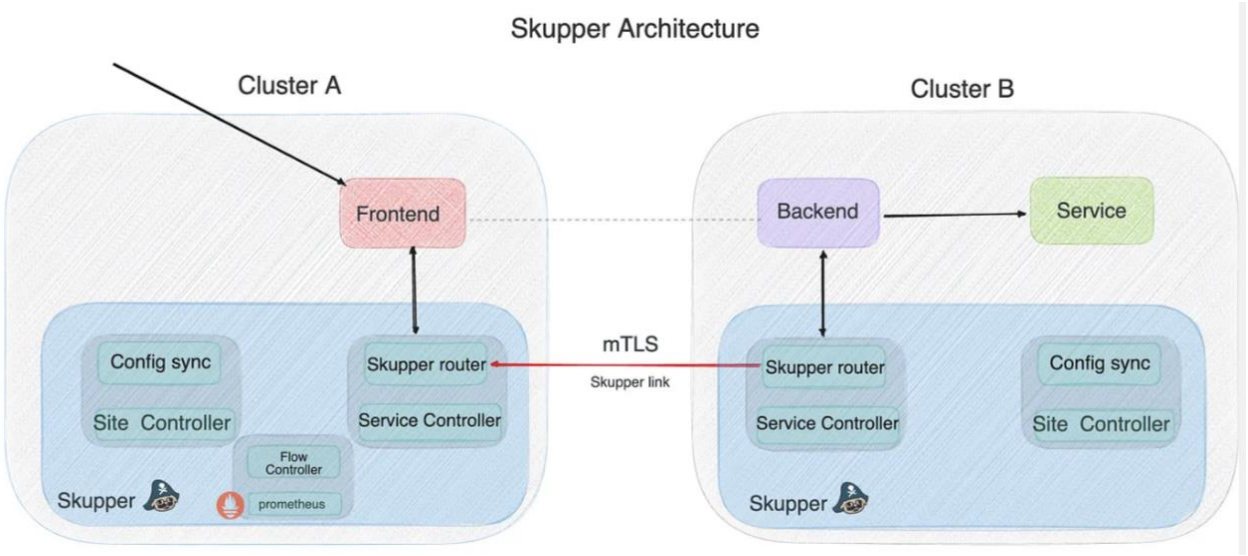

Figure 5: Skupper architecture

Submariner specializes in facilitating secure communication and connectivity between Kubernetes clusters. Administrators can monitor the status of inter-cluster connections, tunnel health and endpoint availability through Prometheus's integration with Submariner. This enables administrators to collect, analyze, and visualize monitoring data effectively. Prometheus scrapes metrics endpoints exposed by Submariner components, such as gateway nodes and controllers, facilitating real-time monitoring of network performance [4].

The current EURECOM's SD-WAN provides overlay monitoring for virtual tunnels established between edge

devices. Metrics such as latency and packet loss of overlay tunnels are exposed through the controller's northbound API.

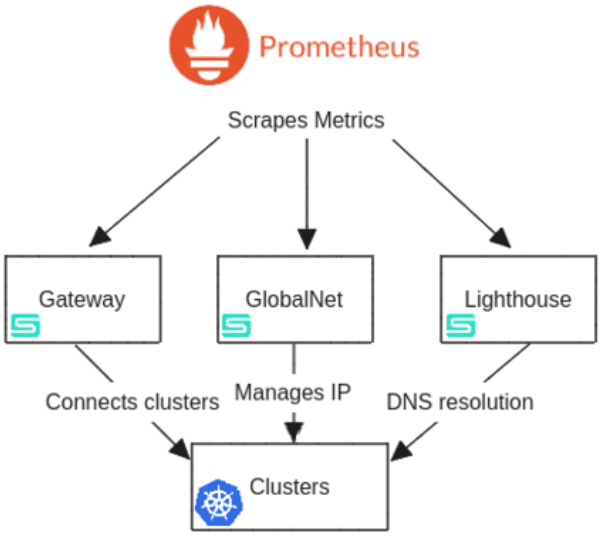

Figure 6: Illustration of submariner capability

4.3.1 State-of-the-Art on Data Collection Interfaces for Computing

In modern computing environments, the challenge of effectively monitoring diverse and dynamic infrastructures is significant, especially in AC³, where the goal is to register any resource to the federation. These environments often consist of a mix of virtual machines, containers, and cloud services, each generating vast amounts of data that must be managed and analyzed to ensure optimal performance and availability. Traditional monitoring solutions can struggle with the scalability and flexibility required to handle such heterogeneous and fluid systems efficiently. An example is Nagios[3] or Zabbix[4], that are designed around a central server architecture. In such (monolithic) setups, all monitored data must be sent to a single server that performs data processing and alert management. This centralization can become a bottleneck as the number of endpoints and the volume of data increase [5]. For instance, if an organization scales out its infrastructure rapidly, the central server can become overwhelmed with the sheer volume of incoming data, leading to delayed alerts, slow processing times, and potential data loss. Additionally, scaling these systems often requires significant manual intervention to add resources and optimize performance, which can be resource-intensive and error-prone. Another aspect that should be considered is that many older systems are configured to collect a standard set of metrics with limited ability to customize or extend what is monitored without significant effort [6]. For instance, tools like HP OpenView provide a range of metrics and monitoring capabilities, which might not cover all the metrics needed in a more modern, microservices-based application. Also, it fails to consider performance indicators as well as constraints/benchmarking evaluations that the resources may have [7].

An approach could be a decentralized system where each component or service independently monitors its metrics and reports back to a central aggregator. Although this can reduce the load on a single point and improve resilience, it can also complicate the configuration and maintenance of the overall system. Another approach is to employ a push-based model, where data is sent to the monitoring system in real-time. While this approach

[3] https://www.nagios.org/
[4] https://www.zabbix.com/

provides immediate data availability, it can overwhelm the monitoring system during peak times with high data inflow. Given the diversity of resource types and the unpredictable nature of their interconnections, the performance of a monitoring infrastructure can vary greatly. In such a landscape, Prometheus emerges as an excellent solution that leverages the strengths of different monitoring approaches while addressing their limitations. Prometheus utilizes a pull-based model to actively gather metrics from pre-configured endpoints at set intervals. This approach ensures that data collection remains manageable and current, while also minimizing the risk of data overload that is often associated with push-based systems. Furthermore, Prometheus is designed for high scalability, featuring robust service discovery that automatically adapts to changes within the environment, such as when endpoints are added or removed. Additionally, its storage system is adept at handling large volumes of time series data, which is particularly critical for complex use cases like those of AC³. Moreover, the Prometheus Query Language (PromQL) offers extensive capabilities for efficient data processing and aggregation, enhancing the monitoring system's overall functionality and responsiveness.

4.3.1.1 Data Model using Computing Resources

The data model and the corresponding monitoring setup provide a robust foundation for maintaining insight into the performance and health of a cluster, ensuring that the usage of computing resources is optimized, and issues can be proactively addressed even in scenarios where the infrastructure has edge or far edge nodes.

In AC³ the objective is not only to develop a generic model, but it is also important to create an effective one for monitoring the infrastructure. This process should begin with a clear understanding of what is needed to be monitored in order to ensure the health, performance, and efficiency of a cluster. Monitoring objectives fall into the following broad categories:

- **Resource Usage:** Tracking the consumption of CPU, memory, storage, and network resources is fundamental. These metrics help ensure that the cluster operates within its capacity and can forecast through the AC³ components when additional resources might be needed.
- **Performance Metrics:** Measuring response times, throughput, error rates, and other performance indicators is essential to ensure that applications running on Kubernetes meet their SLA and provide a good user experience.
- **Health and Availability:** Regular checks on the health of pods, services, and nodes help in identifying problems early. This includes monitoring the status of pods (e.g., running, waiting, or crashed) and nodes (e.g., ready, disk pressure, memory pressure), as well as the success rates of liveness and readiness probes.
- **Cluster State and Scalability:** Understanding the state of the cluster involves monitoring the status and number of nodes, the distribution and health of workloads, and the effectiveness of scaling mechanisms. It also includes tracking the deployment statuses to ensure that the desired state matches the actual state, which is critical for deployments and stateful sets.
- **System Events and Logs:** Collecting and analyzing logs and events can provide deeper insights into what is happening within the cluster. This includes understanding pod scheduling decisions, system errors, and significant state changes.

Another layer of abstraction is the identification of key metrics critical for monitoring and managing clusters effectively. These metrics, essential for understanding both the current state and historical trends of cluster resources (important input for the AI-based LCM), are categorized into four groups: Pod Metrics, Node Metrics, Deployment and Stateful Set Metrics, and Resource Usage Metrics. Each category serves a unique purpose, providing insights that help in proactive management and optimization of the cluster.

The first characteristic is the Pod Metrics that are crucial for tracking the operational status and environment of the smallest deployable units in Kubernetes. Metrics such as kube_pod_info provide valuable metadata, including namespace and pod-specific labels, which are instrumental for contextual analysis across the cluster. The kube_pod_status_phase metric indicates the lifecycle phase of each pod (Pending, Running, Succeeded, Failed, Unknown), offering immediate visibility into potential disruptions or inefficiencies. Resource allocation metrics, kube_pod_container_resource_requests_cpu_cores and kube_pod_container_resource_limits_memory_bytes, detail the CPU and memory specifications requested and limited per container, aiding in resource management and capacity planning.

Other key characteristic is the Node Metrics that give a broader perspective on the physical or virtual machines hosting the pods. kube_node_status_allocatable reveals the resources that are available for pod allocation, crucial for understanding resource availability and constraints. kube_node_status_capacity compares these figures against the total resources of a node, highlighting the consumption by system processes and the overall resource capacity. Final characteristic in this category is the kube_node_status_condition that provides real-time status conditions of nodes such as readiness and any existing issues, which is vital for maintaining the health and reliability of the cluster infrastructure.

The third group is the Deployment and Stateful Set Metrics, where the kube_deployment_status_replicas and the kube_statefulset_replicas can be used where these metrics focus on the management and scaling aspects of applications. These metrics track the number of replicas deployed against the desired count, ensuring that the application scales correctly in response to demand and maintains resilience through adequate replication.

Lastly, Resource Usage Metrics examine specifics of resource consumption at the container level, facilitated by cAdvisor integration with Prometheus. Metrics that are being monitored are container_cpu_usage_seconds_total and container_memory_usage_bytes that provide granular insights into the resource usage of individual containers. This data is invaluable for troubleshooting, optimizing resource utilization, and enforcing resource usage policies to prevent overutilization or underutilization.

4.3.1.2 Data Model with Information on the Energy Type

For energy type metrics, Kepler Exporter[5] can be used. Kepler which stands for Kubernetes-based Efficient Power Level Exporter is designed to monitor and improve the energy efficiency of Kubernetes clusters. It allows us to collect and import energy consumption metrics. It exposes statistics from an application running in a Kubernetes cluster in a Prometheus-friendly format that can be scraped by any database that understands this format.

Kepler exports several metrics to Prometheus, which are categorized by the type of energy consumed:

- **CPU Energy:** *kepler_container_core_joules_total tracks* the energy used by the CPU cores. This measures the total energy consumption on CPU cores that a certain container has used.
- **DRAM Energy:** kepler_container_dram_joules_total records the energy consumed by the DRAM.

Information obtained from Kepler's exporter can be used for profiling the resources with the Resource Profile Schema described in Section 4.2.2.1. For example, we need to convert the Joule-based metrics from Kepler (such as *kepler_container_core_joules_total*) into watts using Prometheus queries. We can also use the rate() function to calculate energy consumption per second (watts).

4.4 Resource Discovery, Unified Resource Discovery Communication

In today's highly distributed network environments, efficient management of resources across the CECC is imperative. The AC[3] project should address this need through innovative and sophisticated resource discovery

[5] https://sustainable-computing.io

mechanisms and unified resource communication strategies. These mechanisms are integral to optimizing resource allocation, enhancing system responsiveness, and ensuring robustness across diverse infrastructures, including cloud, edge, and far edge layers.

Resource Broker module should comply with the Resource Profile Schema discribed in Section 4.2.2.1. In order to achive this, Resource Broker will provide to AI-based LCM module a JSON schema with the same format. The major challenge is to unify all data provided from resource discovery of different computing entities such as cloud providers or custom provisioned data centers. To address this challenge a simple yet highly effective solution would be to employ Kubernetes as the base building block where application will be deployed.

4.4.1 Resource Discovery

Having made the previous assumption about employing Kubernetes for applications to be deployed, state-of-the-art tools can be used that have the added benefit of being production tested.

Every time a new LMS is being provisioned, it will communicate its existence and the initial available resources to the Resource Discovery mechanism through an API call, also providing information on the real-time Resource Exposure endpoint.

Resource Discovery module will communicate with the Resource Exposure of each LMS site, Kurbernetes clusters, in order to get information about the available resources. Each LMS's Resource Exposure should have a Kubernetes Metrics server, which will be the Resource Exposure endpoint. There are several metrics that can be monitored through the Metrics server, such as:

- **Cluster state metrics:** These metrics focus on the health and availability of Kubernetes items (nodes, pods, etc.)
- **Resource metrics:** These metrics allow you to understand whether the cluster can handle its workloads and whether it can handle new loads. It is possible to track the use of resources at different levels of the cluster. This group includes the following metrics: memory requests, memory limits, allocatable memory, memory utilization, CPU requests, CPU limits, allocatable CPU, CPU utilization, and disk utilization.

Resource Discovery will scrape all the available Resource Exposure endpoints of all LMS instances in order and then will communicate this information to the Resource Broker for transforming the data to the JSON format that was described previously in Section 4.2.2.1.

5 AI/ML Models for Prediction and Resource Management

5.1 State-of-the-Art

The integration of ML and DL into resource management has significantly transformed cloud and edge computing. These advancements are highlighted through a comprehensive analysis of research contributions, including studies on ML-centric resource management in cloud computing and the innovative applications of DRL in network resource optimization. In the domain of cloud computing, the shift towards ML-based dynamic resource management methods is primarily aimed at accommodating the variability and complexity of cloud workloads. In [8], the authors show the importance of developing advanced ML models to improve the accuracy of workload predictions, optimize the utilization of resources in virtual machines (VMs), and realize energy-efficient data center operations. The future of cloud computing resource management, as suggested by [8], lies in the creation of sophisticated ML models and adaptive management strategies that can accurately predict and efficiently allocate resources. The adoption of ML and DL offers promising solutions by enabling intelligent prediction, adaptation, and optimization of network operations. This includes resource allocation, power distribution, and traffic management, automating decision-making processes to ensure more efficient, reliable, and energy-saving cloud and edge computing. The emergence of DRL as a powerful tool in resource management shows its ability to dynamically adjust and optimize resource allocation based on changing conditions and demands. Authors in [9] give an example of the application of DRL in managing system and network resources by translating complex allocation tasks into a learning problem. This approach not only matches the performance of state-of-the-art methods but also intuitively learns effective strategies through direct experience, marking a paradigm shift towards more flexible and efficient resource management methods. However, despite these advancements, several challenges remain, including the need for explainability of DRL models, scalability, decision-making complexity, and adaptability to different network environments. Addressing these challenges requires further research focused on developing sophisticated ML models that can provide accurate predictions, consider a wider range of resources, and implement adaptive management strategies. The application of ML and DRL in resource management represents a significant step towards more adaptive and intelligent systems capable of handling the complexities of modern computing environments. By enhancing the efficiency, adaptability, and performance of cloud and edge computing, ML-centric approaches are paving the way for the next generation of computing networks and systems. As the field continues to evolve, the development and integration of advanced ML models and solutions promise to unlock new levels of efficiency and innovation.

5.2 Data, Computing and Networking Metrics

In the context of managing resources within the CECCM system, AI and ML models will try to ensure efficient resource utilization. To achieve this, we rely on a comprehensive set of Key Performance Indicators (KPIs) that provide insights into various aspects of system performance, reliability, and security. Table 2 outlines the KPIs used in our AI/ML models for resource management, explaining their significance and how they contribute to the overall optimization process.

Each KPI serves a specific purpose, enabling us to build sophisticated AI/ML models that predict future resource requirements, identify potential bottlenecks, and optimize resource allocation dynamically. By continuously monitoring these metrics, our models can proactively adjust resource distribution, ensuring that applications run smoothly and efficiently even under varying load conditions. This proactive approach not only enhances user experience by minimizing latency and downtime but also optimizes the use of underlying infrastructure, reducing operational costs and improving overall system sustainability.

Table 2 provides a list of the actual counters that can be used per category.

Table 2: List of the actual counters

Counter	Categories
Response Time	response_time response_time_avg response_time_max response_time_min
Error Rate	error_rate error_count error_ratio errors_per_second
Transaction Throughput	transaction_throughput transactions_per_second transactions_rate
Connection Persistence	connection_persistence persistent_connections connection_persistence_rate
Transaction Continuity	transaction_continuity transaction_count transactions_per_second transaction_success_rate
HTTP Requests	http_request_total http_request_rate
HTTP Response	http_response_time http_response_code_2xx_count http_response_code_3xx_count http_response_code_4xx_count http_response_code_5xx_count
SSL Connections	ssl_connections ssl_total_connections

	ssl_current_connections
SSL Handshake Time	ssl_handshake_time ssl_handshake_duration ssl_handshake_latency
Database Query Times	db_query_time db_query_response_time db_query_duration
Cache Hit Rates	cache_hit_rate cache_hit_ratio cache_hits cache_misses
Throughput	throughput total_throughput network_throughput data_throughput
Network Latency	network_latency latency_avg latency_max latency_min
Security Metrics	security_events_total security_events_rate security_violations security_threats_detected security_attacks_prevented
Load Balancer Metrics	load_balancer_throughput load_balancer_errors load_balancer_response_time load_balancer_server_health load_balancer_rule_hits

5.3 AI/ML to Predict Infrastructure Usage

5.3.1 State-of-the-Art

Infrastructure usage in the CECC refers to the utilization of various computing resources at the edge or cloud level. These resources can range from compute resources to storage resources to network resources. Most existing work in predicting infrastructure usage focuses on computing resources. Predicting infrastructure usage, particularly computing resources, has typically followed two directions. The first direction involves predicting the workload and then inferring the cloud computing resource usage, such as the CPU or memory, while the second direction involves considering the cloud computing user pattern to predict computing resource usage directly. Several approaches to workload prediction in the context of cloud resources have been explored. Some of these approaches employ classical time series forecasting mechanisms such as autoregressive integrated moving average (ARIMA) [10]. However, these approaches have shown limited learning capability compared to more advanced ML models.

ML models have been extensively utilized for workload prediction in the context of the cloud, as demonstrated in [11], where the authors trained a regression model to forecast the resources (number of VMs) required to meet a specific response time, SLO, for the predicted workload. They specifically evaluated various state-of-the-art regression methods, including Elastic Net (EN), Linear Regression (LR), Polynomial Regression (PR), and Decision Tree (DT), to predict the workload. In [12], a workload prediction approach is developed using Support Vector Regression (SVR) and web server workload data to evaluate the required resources. Their ultimate goal was to minimize latency while reducing infrastructure costs and energy consumption. Another approach, called CloudInsight, is developed in [13], where the objective is to create a cloud workload prediction framework leveraging multiple workload predictors to create an ensemble model to improve the accuracy of the predicted cloud workload. The specificity of the ensemble model is that it is periodically optimized to address sudden changes in workload. Similarly, [14] developed a workload prediction mechanism based on ensemble methods, demonstrating the superior performance of their approach compared to K Nearest Neighbor (KNN), Neural Network (NN), DT, Support Vector Machine (SVM), and Naïve Bayes (NB).

A more granular approach to predicting cloud resource usage with ML is addressed in [15], where the prediction is tailored to the task and resource. Along the same vein of granularity, [16] presents a prediction model that provides short- and long-term cloud resource usage predictions, enabling the proposed solution to adapt to different load characteristics and temporary and permanent usage changes. In [17], a VM consolidation approach is proposed that takes into account both current and future resource utilization. Their approach uses a regression-based model to approximate the future CPU and memory utilization of virtual machines (VMs) and physical machines (PMs). They investigate the effectiveness of virtual and physical resource utilization prediction in VM consolidation performance using Google cluster and PlanetLab real workload traces. Their experimental results show improvements over other heuristic and meta-heuristic algorithms in reducing energy consumption, the number of VM migrations, and the number of SLA violations.

In addition to traditional ML methods, DL techniques, particularly those using Long Short-Term Memory (LSTM), have been explored for predicting cloud resource usage. For instance, [18] investigates a model selection approach that captures common resource usage patterns across different tasks within a job. They employ a set of LSTM models trained at the job level, allowing them to select the appropriate model during inference for predicting resource usage accurately. In a thought-provoking study, [19] questions the necessity of complex machine learning models for cloud resource prediction. They find that LSTM models achieve high prediction accuracy even on unseen data. However, upon closer examination, they notice that the predicted values closely resemble the original data shifted by one-time step into the future. Other studies, such as ([20], [21], [22], [23], [24], [25]), have also explored various DL approaches for cloud resource prediction. These include methods based

on LSTM, Bidirectional LSTM (Bi-LSTM), Gated Recurrent Unit (GRU), and Convolutional Neural Networks (CNN). These efforts aim to improve the accuracy and efficiency of cloud resource management and optimization.
In recent years, advancements in DL, particularly the transformer architecture [26], have facilitated the development of sophisticated techniques for workload and resource usage prediction. In [27], the authors propose a workload prediction approach that decomposes time series data into trend items, period items, and residuals. They employ quantile regression and exponential smoothing for prediction and additionally explore transformer-based prediction models in situations with ample data. Similarly, [28] introduces a deep attentive periodic model based on the transformer for multi-dimensional, multi-horizon workload prediction. This model delivers precise and reliable workload information crucial for scaling operations. Notably, their approach features a lightweight periodicity extractor capturing inherent workload seasonality and a periodicity attention module learning periodic workload dependencies. In [29], the authors develop an elaborate workload prediction model predicting the workload associated with each microservice using a spatio-temporal GNN. They employ various approaches for temporal time series data prediction and highlight the advantages of the transformer over methods like DT Regressor, SVM, and LR. Despite the efficacy of these methods, it is worth noting that they lack explainability in their predictions.

5.3.1.1 Beyond State-of-the-Art

The existing approaches for infrastructure usage prediction are either based on classical time series forecasting approaches, ML or DL methods. They also consider the recent advances in DL, such as the transformer, to predict infrastructure usage. However, they do not consider XAI models for infrastructure usage prediction. The application of XAI approaches to time series data has attracted an important consideration [30].

5.3.2 XAI-Enabled Fine Granular Resources Autoscaler

In this section, we describe a Zero-touch Service Management (ZSM) framework featuring a fine-granular resource scaler algorithm to run microservices in a cloud-native environment optimally, proposed in [31]. The scaler algorithm relies on ML models to predict the performances of the run microservice. When a service degradation is detected, XAI algorithms are used to interpret the ML prediction and deduce which features led to that bad performance. More specifically, the framework relies first on an ML algorithm based on eXtreme Gradient Boosting (XGBoost) [32] to predict any violations related to the performance of running applications. The characterization of the application performance is done using the application response time metric. The trained ML model considers many features related to CPU and memory, namely CPU usage, CPU limit, memory usage, and memory limit. Parallelly, an XAI algorithm is run, namely SHAP [33], to deduce the most important features that yield such violations using ML outputs. By knowing the root cause of the performance violation, the autoscaler algorithm scales the CPU, memory, or both. Regarding the scale-down process, we consider a threshold-based approach for CPU and memory, in which a scale-down is possible. But, in order to avoid a ping-pong effect (repetitive scale down and scale up), we also consider a stabilization period after a scale-up where a scale-down process is not allowed.

The vertical autoscaling framework can be combined with existing horizontal scaling mechanisms in order to achieve both vertical and horizontal resource autoscaling.

5.3.2.1 *Design and Specification of the Autoscaler Framework*

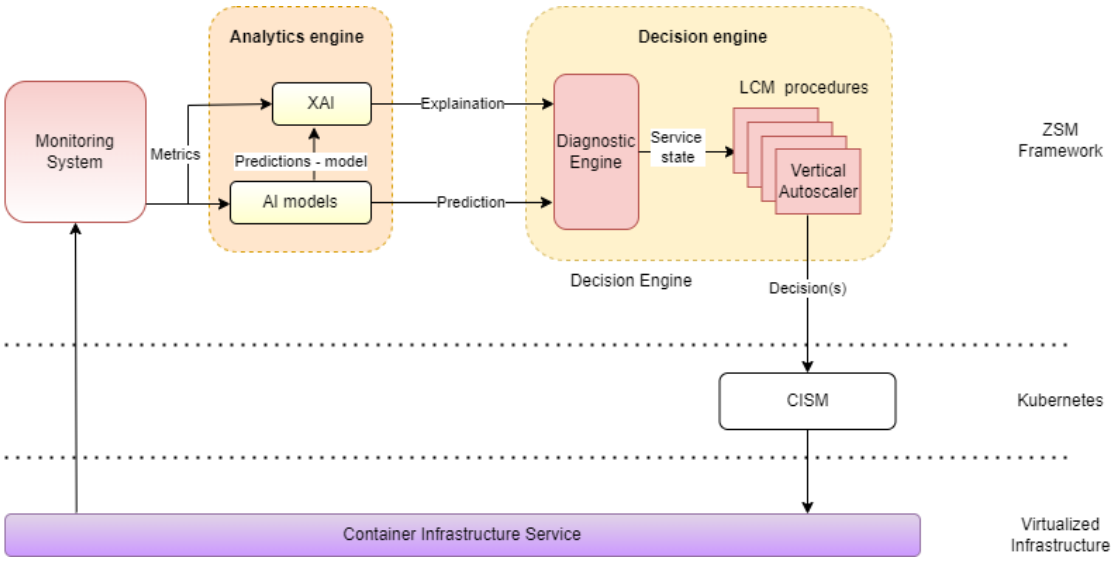

Figure 7: Zero-touch service management framework architecture

Figure 7 illustrates a generic architecture of the proposed ZSM framework. We assume that all microservices run in a cloud-native environment. The figure separates between the closed-control loop components described in the preceding paragraph and the virtualized infrastructure and its manager. The latter is known as the assisted system based on ETSI Experiential Networked Intelligence (ENI) group's notation [34]. According to ETSI cloud-native report [35], the cloud-native equivalent of a hypervisor is Container Infrastructure Service (CIS), which provides all the runtime infrastructural dependencies for one or more container virtualization technologies. In contrast, Container Infrastructure Service Management (CISM) is a cloud-native equivalent of Virtualized Infrastructure Manager (VIM). Technologically speaking, CSIM may correspond to Kubernetes. Regarding the closed-control loop, the Monitoring System (MS) monitors the KPI from CIS regarding the container's resource usage, such as CPU and memory consumption. In our case, we extracted information regarding computing resource consumption (CPU, memory) that the Analytics Engine (AE) will use to predict the performance of the microservice at the service level. Here, we are interested in predicting QoS as perceived by the end-users. In the context of a web server, the metric reflecting the QoS can be the response time, i.e., the time a web server takes to answer a client request. Usually, high service time means the server is overloaded and cannot handle the requests in a bounded time, hence degrading the user's quality of experience. AE runs the trained ML model along with the XAI algorithm to predict whether the response time corresponds to service degradation. The XAI module uses both the collected KPI as well as the ML prediction to provide an explanation. Both the explanation and the prediction are transmitted to the Decision Engine (DE) and, more precisely, to the Diagnostic Engine module (Figure 7). The latter uses the output of the AI model responsible for detecting whether the application response time is appropriate or not. It also receives explanations from the XAI module about inference. The explanation gives the contribution of the features to the model output, which means that if the model detects a high response time occurrence (i.e., QoS degradation), the XAI output indicates the contribution of the application's resource features in this result. These characteristics are related to either CPU usage or memory usage. The Diagnostic Engine then detects the element that caused the high response time. This information is then passed to the vertical autoscaler algorithm, which decides on which resource to scale, hence performing a fine-granular scaling rather than blindly scaling both CPU and memory. It is worth noting that the autoscaler decision is enforced using the northbound API exposed by CISM that allows updating the resources dedicated to the container running the application by modifying the application's controller object of Kubernetes. Afterward,

the Kubernetes controller will roll out a new instance with the new resources definition and delete the old instance.

Regarding the scaling down process, we use a stabilization period after a scale-up in which scaling down will not be performed in order to avoid resource scaling oscillations, i.e., the autoscaler performs one action and, after a short period, performs the opposite action. If no performance drop has been detected during the last few seconds of the stabilization period, a scaling down is possible. To perform this operation, we rely on historical data on resource usage. For memory, if during the last stabilization period, the maximum memory usage was under a chosen percentage, then a scaling down of memory resources is possible. For CPU, if the mean CPU usage during the last stabilization period was less than a certain percentage, then a scale down of CPU resources can be performed. It is worth noting that the scaling down process has no impact on the granularity of the resource allocation. Indeed, when scaling down an application, the resources are released and can be used by another running application instance.

5.3.2.1.1 Analytical Engine (AE)

The analytical engine is responsible for analyzing the microservices' performance and detecting QoS degradation. It is composed of two main components: (1) the AI model, which is based on XGBoost, to predict the latency performance of a microservice. (2) The XAI model interprets the output of the AI model using SHAP.

ML Training

Considering the collected dataset, we can observe that the resources allocated to the application and the relative usage of resources are related to the performance of the application. First, the allocated resources show the limit of performance; the application with fewer available resources will perform worse. Second, the relative resource utilization indicates the possibility of the occurrence of high response times, which means that the degradation of the application performance is more likely to occur when the resource utilization approaches the limit allocated to the application. Therefore, we implement an ML model using the XGBoost classifier to detect performance deteriorations of the application. The model uses resource usage and limits information that can be collected on the running applications via MS. XGBoost is a scalable ML system for tree boosting. It implements the gradient-boosted trees algorithm, a supervised learning algorithm that can be used for regression or classification tasks. We train the XGBoost classifier to detect the application's performance drop based on resource usage patterns. To train the XGBoost classifier on the web server's dataset, we label the dataset's lines as QoS respected or QoS not respected when the response time is lower or higher than a threshold, respectively. Finally, the model gets the following information as input: memory limit, memory usage, CPU limit, CPU usage, relative CPU, and relative memory. Those metrics can be collected for all the running workloads via MS during runtime. Based on the label and the resource usage, the model classifies the performance of the application, using the resource consumption of the workload, into respecting QoS or not based on the resource usage and limit. During training, we compared several classification algorithms: K-Nearest Neighbors (KNN) classifier, Artificial Neural Network (ANN) classifier, Logistic Regression (LR), Random Forests (RF), and XGBoost classifier. The XGBoost model was selected based on the classification report by comparing the precision and recall for class 0, which represents the performance degradation of the service. The model's accuracy was 0.95, and the precision and recall for both classes (0 for Qos not respected and 1 for QoS respected) were respectively 0.86, 0.74 for class 0, and 0.97, 0.99 for class 1.

XAI

The second element of the AE is the XAI module, which is responsible for interpreting the output of the AI model. Several XAI techniques exist and can be classified into global or local explanation techniques. Global explanation techniques, such as SHAP, are applied to obtain the general behavior of a model by attempting to explain the whole logic of a model by inspecting its structure. On the other hand, local explanation techniques, such as SHAP

and LIME [36], tackle explainability by segmenting the solution space and giving explanations to less complex solution subspaces that are relevant to the whole model. These explanations can be formed through techniques with the differentiating property that only explain part of the whole system's functioning. The XAI module of the AE relies on the local explanation method based on SHAP to compute the scores of the features contributing to the model's output. The module's output is the contribution score values of the features to the output. Figure 8 represents a visualization of an output of the SHAP method for an ML prediction. The negative values indicate that the feature pushes the model's output towards the output 0. while the positive values signify that the feature pushes the output of the model towards the positive output 1.

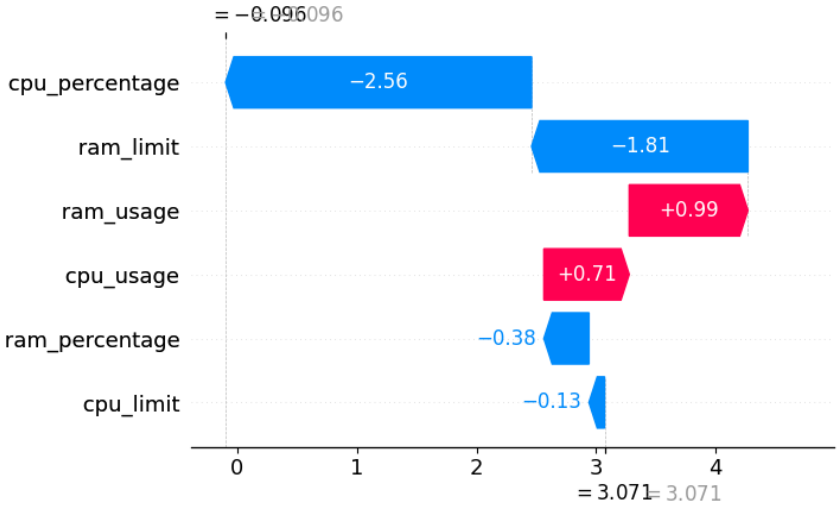

Figure 8: Shapley values provided by the SHAP method

For this inference, the XAI module reports the following Shapley values or scores of the features, ordered by decreasing contribution to the model output: "CPU_percentage" -2.56, meaning that the "CPU_percentage" value pushed the model towards the output 0 (SLA not respected) with a score of 2.56, the second affecting feature is "RAM_limit" with a score of -1.81, the next contributing feature is "RAM_usage" with a score of +0.99 meaning that this feature pushed the model towards the output 1 (SLA respected) with a score of 0.99; for the less impacting features, the XAI module indicates "CPU_usage" +0.71, "RAM_percentage" -0.38, and "CPU_limit" -0.13.

Afterward, the selection of the resources to scale up is made at the DE level. This is done by comparing the weighted sum of the contribution of the features related to CPU resources with the weighted sum of the contribution of resources related to memory resources. We can deduce from the previous scores that the combined score of CPU-related features is -1.98 and memory -1.2, meaning that CPU-related features have more influence on the model decision. Therefore, the DE decides that the cause of the performance drop is insufficient CPU allocation. This information allows the vertical autoscaler to decide to allocate more CPU resources to the workload.

5.3.2.2 Performance Evaluation

For the performance evaluation, we use two versions of the vertical autoscaler, an XAI-based autoscaler to scale the web server instances resources vertically and one without using the XAI output. The running application is exposed to requests load produced by a test component that uses ApacheBench[6] to make several concurrent

[6] https://httpd.apache.org/

HTTP requests. We refer to a test round as a set of N requests made by C concurrent clients. It is worth recalling that AE runs the ML model to predict QoS degradation. The latter was trained in the dataset related to the performance of web servers under changing configurations. If the model detects degradation, the XAI module is called. The XAI takes as input both the ML output as well as the dataset to return the Shapley (or score) values of the features as a numerical score. Then, the Diagnostic Engine of DE compares the weighted sum of the memory-related features scores with the weighted sum of the CPU-related features scores. This output will allow the autoscaler to decide what type of resources need to be scaled. In case the XAI module is not involved, the autoscaler obtains information about the service's state using only the ML module's output (XGBoost); it has no information about the contribution of the features to the model output. If degradation is detected, both CPU and memory resources are scaled.

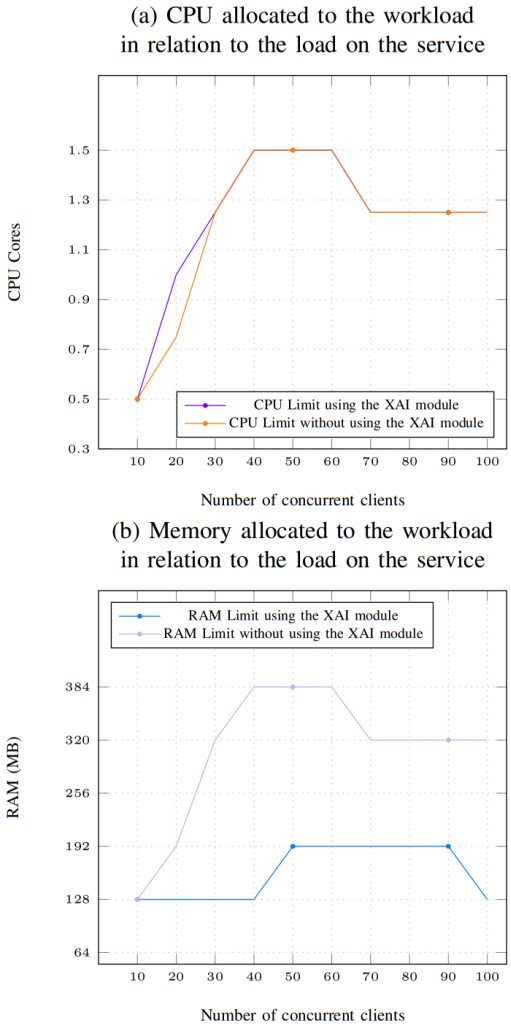

Figure 9: Highest values of CPU and RAM allocated to an instance of the applications in relation to the number of concurrent clients

We performed an extensive test regarding the performances of the two vertical autoscalers. Hence, we deploy 30 applications, and we vary the load to which each application is exposed. The application is first deployed with an initial resource configuration of 0.5 Core of CPU and 128MB of memory.

The number of concurrent clients sending requests to each application varies from 10 to 100, while the number of requests varies from 90 when using 10 concurrent clients to 450 when a concurrency level of 100 is used.

Finally, we perform 100 rounds for each concurrency level. For each application, the pod configuration is reinitialized afterward. Resulting in a total of 300 application instances to be scaled by each autoscaler (30 services exposed to a load varying from 10 to 100). Figure 9 shows the highest amount of CPU allocated to the pod running the application for each load and the highest amount of memory allocated to the pod for both vertical autoscalers. Whereas Figure 10 illustrates the mean response time of the 30 applications for the 100 rounds of requests that each scaled application receives. During the experimentation, the 300 instances that have been scaled using the XAI-based autoscaler employed a total of 184.25 cores of CPU and 29.312 GB of memory, while the 300 instances that have been scaled using the non-XAI-based autoscaler used a total of 188 core of CPU and 48.128 GB of memory. Thus, the percentage of memory gained is 39%, while the percentage of CPU resources gained is 1%.

By comparing the decisions of the two vertical autoscalers we observe that the XAI-based one allocates less memory to the application for all amounts of load. In contrast, it allocates the same or more CPU than the non-XAI-based autoscaler. These results clearly show the fine granularity of the resource allocation achieved by the XAI-based autoscaler, thanks to the ability of the latter to determine the factors that led to performance degradation. Moreover, from the response time plot, we observe that the mean response time of the applications while being managed by both vertical autoscalers is approximately equal, meaning that the allocation of lower resources by the XAI-based autoscaler did not affect the applications' performances.

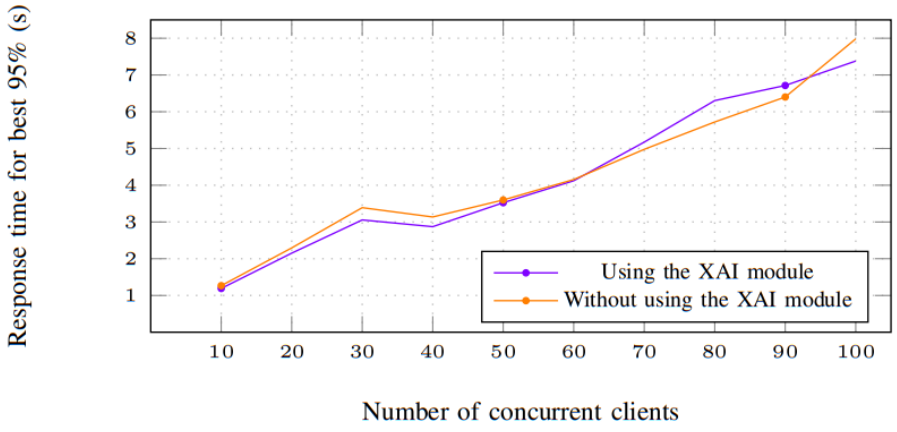

Figure 10: Mean response time in relation to the number of concurrent clients

5.3.3 XAI for Prediciton of Infrastructure Usage

We develop an XAI method based on the transformer for explainable prediction of the latency and determination of the influential feartures, characterized by infrastructure resources such as the CPU and memory. For this purpose, we use the Temporal Fusion Transformer (TFT) [37]. The TFT is an AI model designed for time series forecasting. It combines the transformer architecture [38] with temporal fusion mechanisms to capture temporal patterns in sequential data. It is composed of the multi–head attention mechanism from the transformer with Recurrent Neural Networks (RNNs). The TFT architecture includes an LSTM encoder–decoder and multi–head attention components, primarily composed of gating mechanisms, variable selection networks, and static covariate encoding.

5.3.3.1 Gating in TFT

We adopt the TFT architecture presented in Figure 11 for our experiments. Figure 11 illustrates three building blocks: the variable selection networks, the LSTM encoder-decoder, and the TFT, which combines the GRN with multi-head attention. These blocks receive inputs, including past features, the target variable, and future known

features. They produce forecasts of the latency and their associated quantiles. The rationale of utilizing the latency as our response variable is dictated by the fact that it allows us to have insight into the SLA compliance. In our case, we predict the latency using resources as features, such as the CPU and memory. Then, thanks to the explainability of the approach, we can determine which feature(s) is/are the most influential in the forecasting. With this mechanism, whenever, for example, the predicted latency overtakes a predefined threshold, we can determine at that instant which feature(s) was/were the most influential. Through this process, we can implement corrective measures by acting on the responsible feature and preventing the latency from crossing the threshold, thus being compliant with the SLA. This leads to proactive orchestration of microservices.

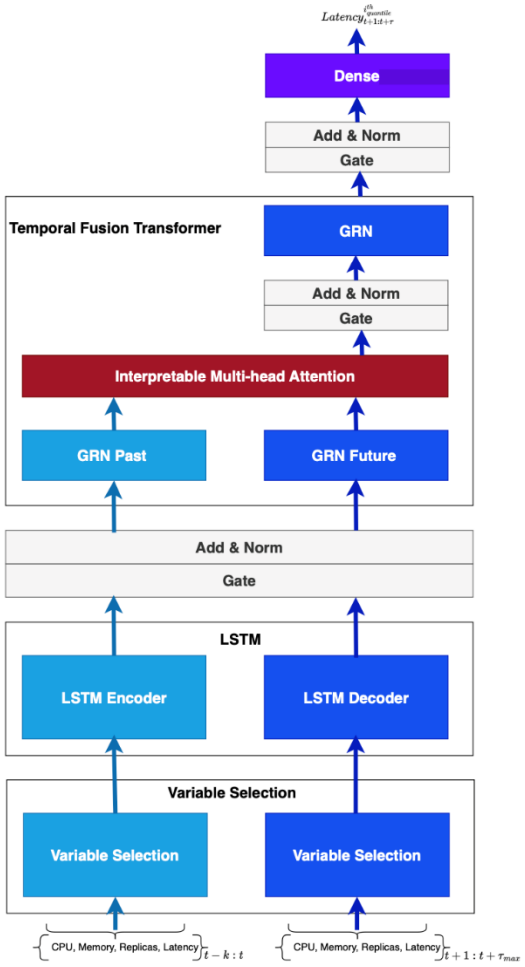

Figure 11: TFT architecture for prediction

The advantage of using TFT comes from the fact that traditional explainability methods for ML and DL approaches, such as LIME and SHAP, are not well-suited for time series data. For instance, LIME constructs independent models for each data point, while SHAP considers features independently for neighboring time steps, disregarding temporal order and critical dependencies between time steps in most situations. This is where the advantages of TFT lie. The fundamental part of the TFT architecture is the Gated Residual Networks (GRN). The information flow through the GRN can be represented as follows

$$GRN(a,\ c)\ =\ LayerNorm\big(a + GLU(\eta_1)\big) \quad (1)$$

$$GLU(\eta_1) = \sigma(W_1\eta_1 + b_1) \odot (W_2\eta_1 + b_2) \quad (2)$$

$$\eta_1 = W_3\eta_2 + b_3 \quad (3)$$

$$\eta_2 = ELU(W_4 a + W_5 c + b_4) \quad (4)$$

a is the input to the GRN, and *c* represents the context vector from static covariates. *W* and *b* are weights and biases optimized during the training phase. *a* and *c* are passed through a dense layer with Exponential Linear Unit (ELU) activation function, followed by a linear layer with dropout. The output from this layer is then fed to the Gated Linear Unit (GLU), where $\odot$ denotes the element–wise product and σ is the sigmoid activation function. The output from the GRN is determined using standard layer normalization of the sum of input *a* and the GLU output (residual connection).

5.3.3.2 Variable Selection Networks in TFT

The role of the variable selection networks in the TFT is to determine the relevant inputs in the time series forecasting process. These networks identify the relevant features that can influence the response variable, thereby providing interpretable forecasting results. The following equations illustrate how the variable selection networks operate in general:

$$\xi_{t,vs}^{j} = GRN(\xi_t^j) \quad (5)$$

$$v_{\chi t} = soft\max\big(GRN(\Xi_t, c_s)\big) \quad (6)$$

$$\xi_{t,vs} = \sum_{j=1}^{m_x} v_{\chi t}^{(j)} \xi_{t,vs}^{j} \quad (7)$$

The first step in the variable selection is to linearly transform all variables at time t to d_{model} dimension. The transformed input for the j^{th}variable is represented by ξ_t^j, where each transformed input is fed to a GRN block, which considers the non–linearities. The output of these blocks is $\xi_{t,vs}^{j}$. Another GRN block which takes as input Ξ_t, the flattened ξ_t^j, and a vector context c_s coming from static covariates, produces a vector $v_{\chi t} \in \mathbb{R}^{m_\chi}$ after passing through a softmax activation function. The variable weights $v_{\chi t} \in \mathbb{R}^{m_\chi}$ are multiplied with the corresponding processed inputs to get the final output of the variable selection networks.

5.3.3.3 Static Covariate Encoding in TFT

The TFT incorporates an approach to integrate static variables, which differs from LSTM and the transformer models. This integration is achieved by generating four context vectors from processed and weighted static variables, produced by the static variable selection networks. Each context vector is then infused into different components of the TFT, where they are utilized for processing. Refer to [37] for more details.

5.3.3.4 Interpretable Multi-Head Attention

The self-attention mechanism of the TFT enables it to learn long-term relationships across different time steps. It is extended in [37] to enhance its explainability capabilities. The attention mechanism is given by

$$Attention(Q, K, V) = A(Q, K)V \quad (8)$$

Where $V \in \mathbb{R}^{N\times d_V}$ are the values, $K \in \mathbb{R}^{N\times d_{attn}}$ are the keys, and $Q \in \mathbb{R}^{N\times d_{attn}}$ are the queries.

A represents a normalization function, where the usual choice is scaled-dot product attention.

$$A(Q,\ K) = soft\max\left(\frac{QK^T}{\sqrt{d_{attn}}}\right) \tag{9}$$

Multi-head attention is usually employed to improve the learning capacities of the standard attention mechanism

$$MultiHead(Q,\ K,\ V) = [H_1,\ \cdots,\ H_{mh}]W_H \tag{10}$$

$$H_h = Attention\left(QW_Q^{(h)},\ KW_K^{(h)},\ VW_V^{(h)}\right) \tag{11}$$

$W_K^{(h)} \in \mathbb{R}^{d_{model} \times d_K}$, $W_Q^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, $W_V^{(h)} \in \mathbb{R}^{d_{model} \times d_V}$

are the head-specific weights for keys, queries, and values, and $W_H \in \mathbb{R}^{(m_H \cdot d_V) \times d_{model}}$ combines linearly outputs concatenated from all heads H_h. The interpretable multi-head attention is given by

$$InterpretableMultiHead(Q,\ K,\ V) = \tilde{H}\ W_H \tag{12}$$

$$\tilde{H} = \tilde{A}\ (Q,\ K)V\ W_V \tag{13}$$

$$= \left\{\frac{1}{H}\ \sum_{h=1}^{m_H} A\left(QW_Q^{(h)},\ KW_K^{(h)}\right)\right\} VW_V \tag{14}$$

$$= \frac{1}{H}\ \sum_{h=1}^{m_H} Attention\left(QW_Q^{(h)},\ KW_K^{(h)},\ VW_V\right) \tag{15}$$

where $W_V \in \mathbb{R}^{d_{model} \times d_V}$ are weights of the values shared across all heads and $W_H \in \mathbb{R}^{d_{atten} \times d_{model}}$ is used for linear mapping.

5.3.3.5 Infrastructure Metrics used and Latency Pattern

We evaluated the TFT explainable approach with real-world data. We specifically focus on the prediction of the latency, with CPU and memory as the infrastructure resources representing our features. We present the CPU, memory, and the mean latency in Figure 12. These metrics are collected at a sampling rate of 5 minutes. To validate our approach, we use different values of the latency, particularly we use different distributions of the latency values. For example, the latency p75 and p95 are considered. As an example, the latency p95 (95th percentile) represents the latency value below which 95% of the measured latency values belong to. Compared, for example, to p99 latency, it is supposed to provide a broader view of the latency distribution. p95 latency is usually used to understand the performance of a system, but it is usually less impacted by extreme outliers compared to p99. If the p95 latency for a microservice is 100 milliseconds, it means that 95% of responses are received in 100 milliseconds or less, but 5% of responses may experience longer latencies. That is one of the reasons why latency is used for monitoring SLA compliance.

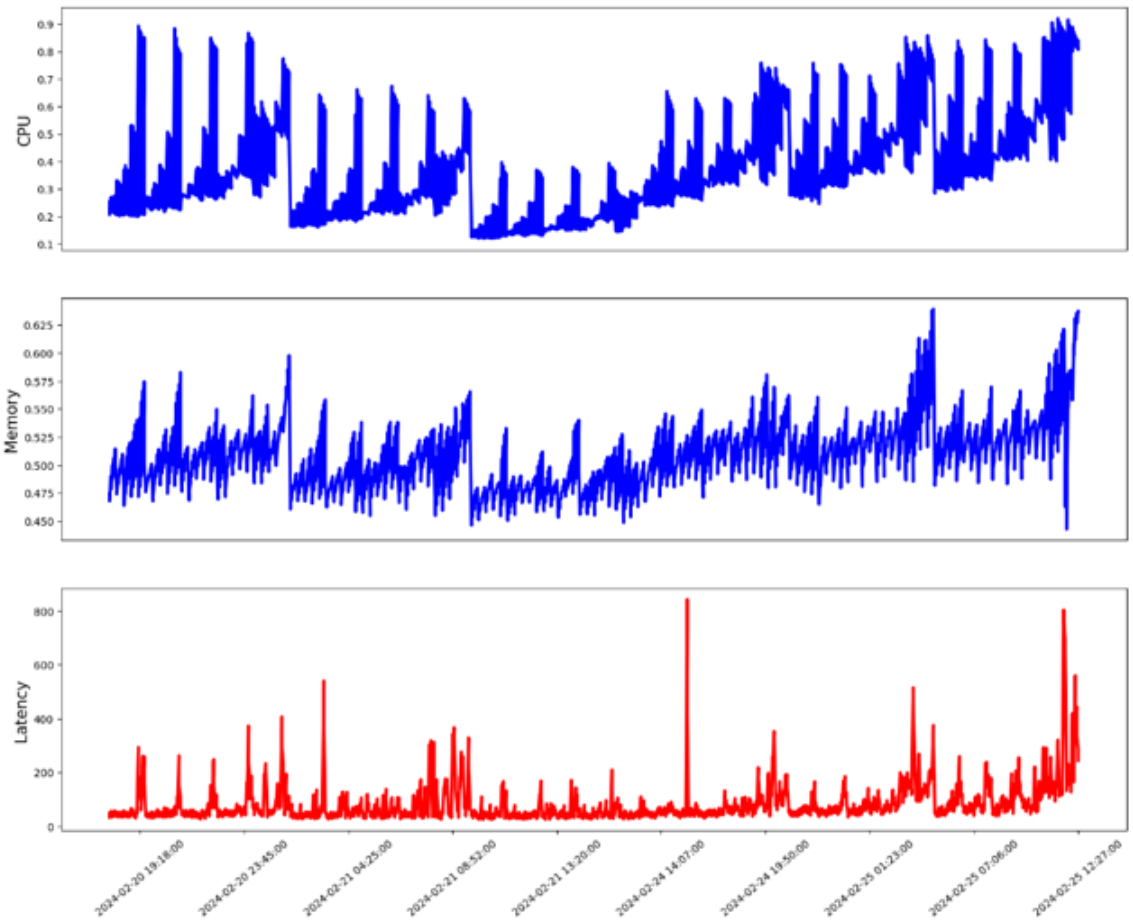

Figure 12: Metrics used

5.3.3.6 Performance evaluation

To build our prediction model for the latency, we utilize features such as the CPU and memory. Our experiments do not rely on any autoregressive features but only on the actual infrastructure resources to forecast future latency. For the TFT model, we include a time index that increments by one for each time step. We standardize each time series separately, ensuring that the values are always positive. To achieve this, we use the EncoderNormalizer, which dynamically scales each encoder sequence during model training. Our model training is conducted using PyTorch Lightnin. The distinctive characteristic of the TFT model is its attention mechanism, which attributes different levels of importance to various points in time during forecasting. This attribute provides explainability to the forecasted results. The TFT is designed for multi–horizon forecasting, meaning that it can forecast future values at multiple time horizons simultaneously; here, for illustrations purposes our multi–horizon is equal to 2. To achieve this, the model incorporates output layers that forecast values for each time horizon of interest. This allows it to generate forecasts for different future points in the latency data. Additionally, we tune parameters such as a batch size of 32, a learning rate of 0.03, and 200 epochs. Our model architecture includes a hidden size of 8, an attention head size of 1, and a dropout rate of 0.1. Furthermore, we set a maximum prediction length of 96, and a maximum encoder length of 168. To evaluate the performance of our model, we utilize a quantile loss function. We use early stopping to avoid overfitting and faster convergence.

5.3.3.7 Forecasting Results

We present the results of the forecasting obtained by applying the TFT to the dataset. The dataset comprises a response variable, latency, and two features, the CPU and memory. Our forecasting horizon is set to 2, resulting in two figures for each forecasted latency. In these figures, the blue curve represents the actual latency, while the orange curve depicts the forecasted latency. The grey curve represents the attention values generated in the Interpretable Multi-head Attention section shown in Figure 11. Additionally, we include the uncertainty estimates associated with the forecasts.

The variable selection process chooses the relevant CPU, memory, and latency values for each time step. It accomplishes this task for both the current and past CPU, memory, and latency metrics. To handle past metrics, an encoder is employed to incorporate the selected features along with an index indicating their relative time. The role of the encoder is to process historical time series data and capture temporal dependencies. It consists of multiple layers of self-attention mechanisms and feedforward neural networks, like the encoder in the transformer model. This encoder encodes infrastructure metrics into a meaningful representation, which then serves as input to the decoder. Additionally, the decoder takes the CPU and memory features for which latency forecasting is desired. In TFT, the decoder primarily generates forecasts of latency. Figure 13 presents the results of the forecasting obtained from the TFT. On the left-hand side, we have the forecasting associated to the 1st horizon of forecasting and on the right-hand side to the 2nd horizon of forecasting. We observe that the losses in both forecasting results are relatively low, and they are given by 4.165 for the 1st horizon of forecasting and by 2.605 for the second horizon of forecasting.

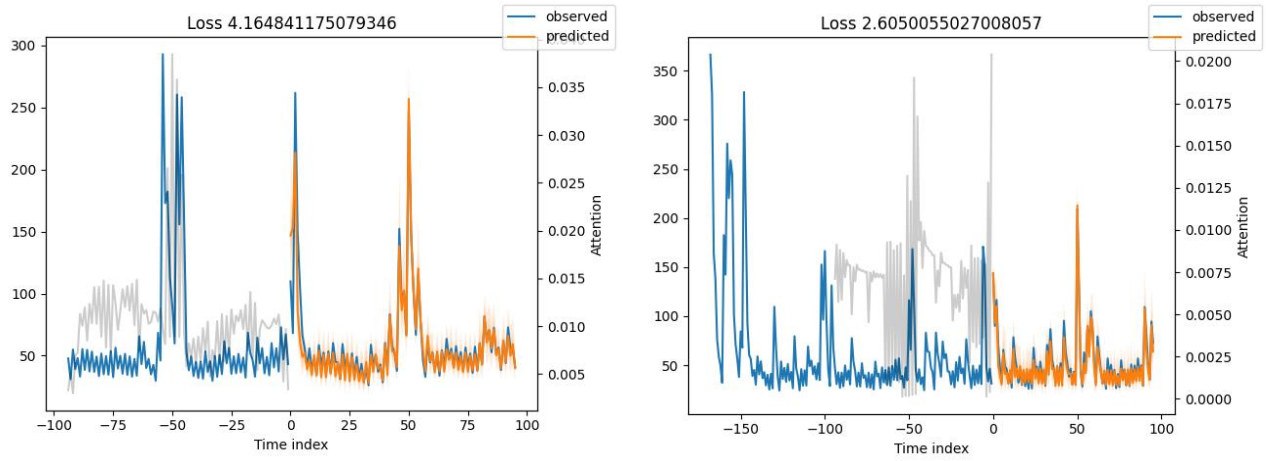

Figure 13: Latency prediction results (mean)

Figure 14 presents the explainability associated to Figure 13, where the left-hand side provides the importance of the features given by the encoder and the right-hand side gives the importance of the features from the decoder. It is worth mentioning that the encoder corresponds to the training phase and the decoder to the forecasting phase. In both cases the consensus is that the most influential feature in the forecasting is the CPU.

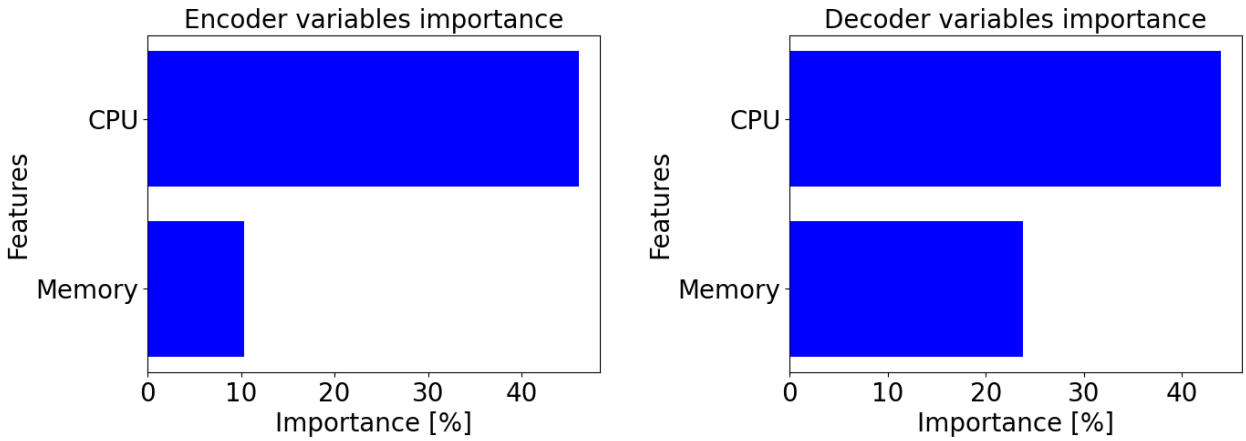

Figure 14: Explainability for the predicted latency (mean)

Figure 15 exhibits the latency forecasting provided by the TFT for the latency p75. Similarly to Figure 13, on the left-hand side we have the forecasting results provided for the 1st horizon of forecasting and on the right-hand side the forecasting provided for the 2nd horizon of forecast. In both cases, we observe that the loss is relatively low. It is given by 1.111 for the 1st horizon of forecasting and by 1.116 for the second horizon forecasting. These two forecasting results demonstrate the accuracy of the TFT.

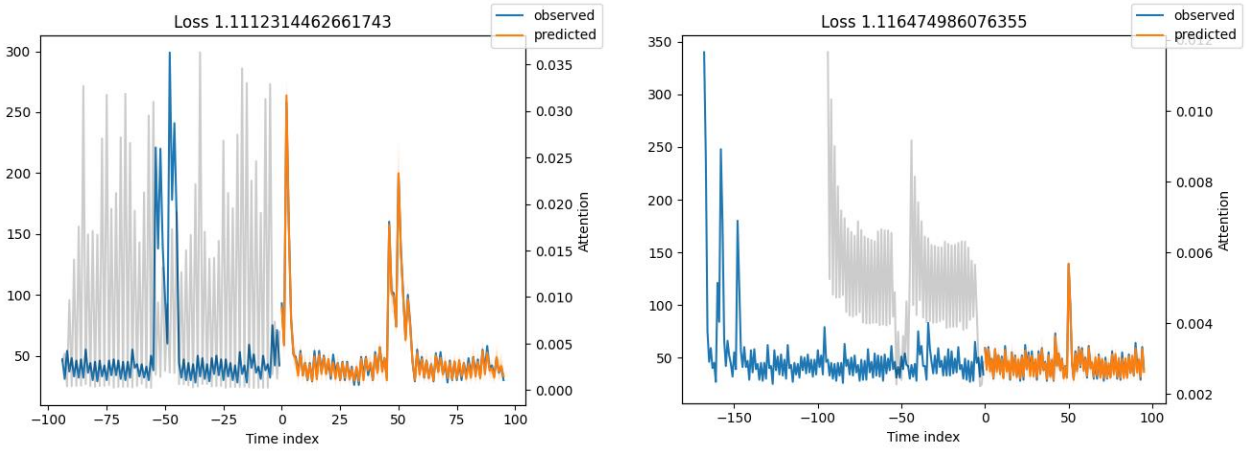

Figure 15: Latency prediction results (p75)

Figure 16 presents the explainability associated to the forecasting results. On the left-hand side the encoder provides the importance of the features in the training phase and on the right-hand side the decoder shows the importance of the features associated to the forecasting. The insight provided by the explainer is that the CPU is the most influential feature.

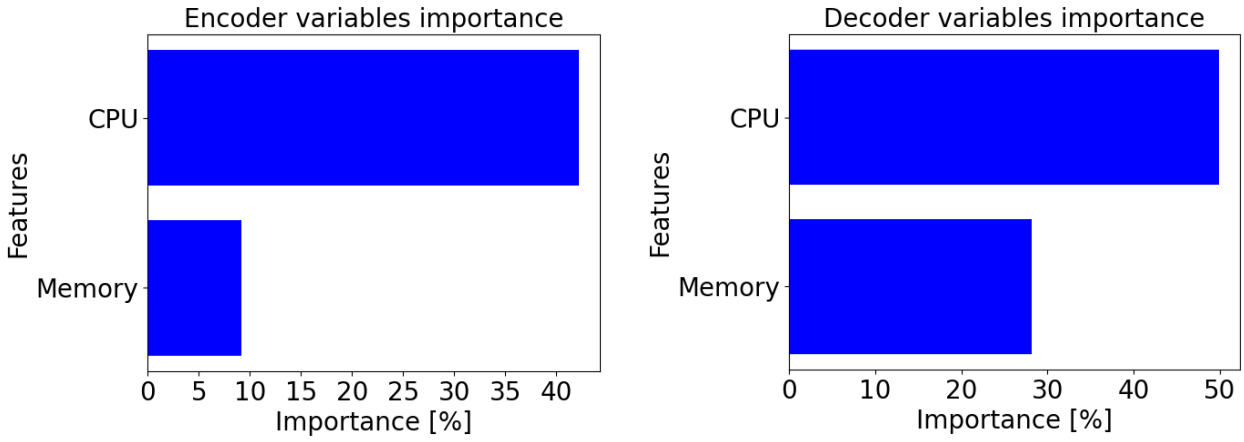

Figure 16: Explainability for the predicted latency (p75)

Figure 17 presents the latency forecasting provided by the TFT for the latency p95. Similarly to Figure 15, on the left-hand side we have the forecasting results provided for the 1st horizon of forecasting and on the right-hand side the forecasting provided for the 2nd horizon of forecasting. In both cases, we observe that the loss is relatively high compared to the 2 previous cases. It is given by 19.272 for the 1st horizon and 14.944 for the second horizon. This is particularly due to the high latency values in this experiment, which is also reflected in the forecasting.

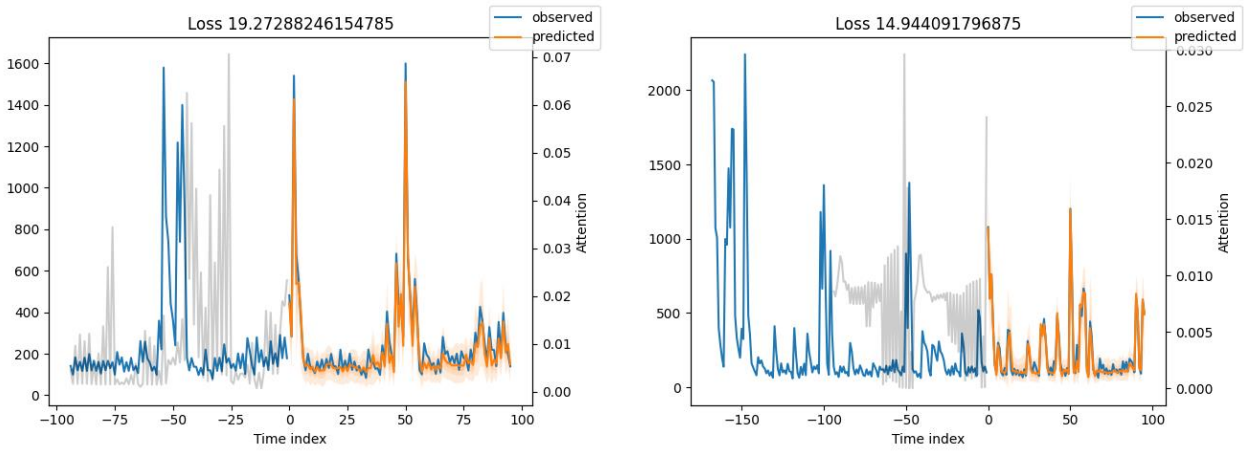

Figure 17: Latency prediction results (p95)

Figure 18 provides the explainability associated to the latency p95. It shows the importance of the CPU feature for both the encoder and the decoder. This conclusion is in line with the previous experiments using various latencies.

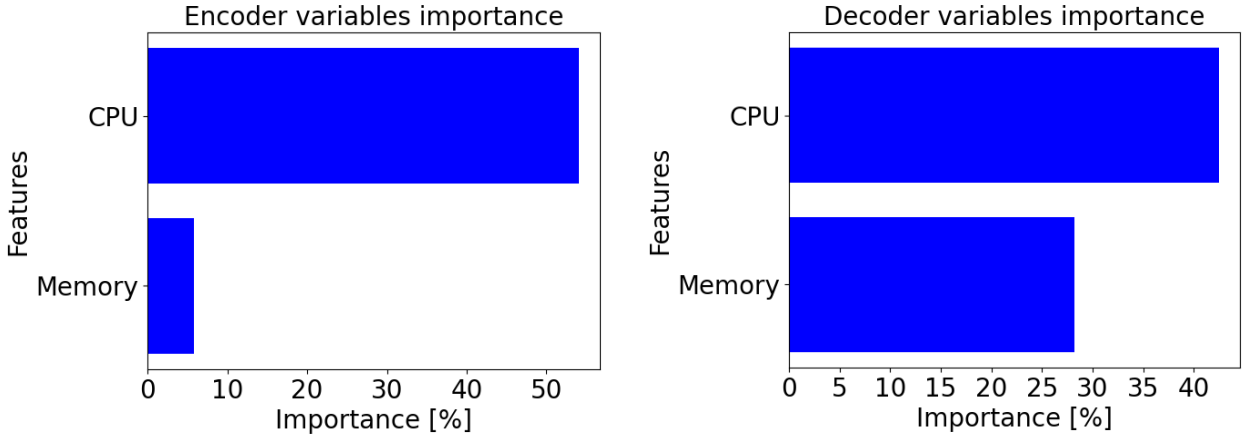

Figure 18: Explainability for the predicted latency (p95)

5.3.3.8 Explainability

One of TFT primary advantages over other DL models is its inherent interpretability, largely attributable to its variable selection and interpretable multi-head attention mechanisms. With TFT, we can determine the significance of CPU and memory in latency prediction, a capability present in both the encoder and decoder components. Across various experiments the consensus is that the CPU emerges as the most influential feature in latency predictions. Across the figures presenting the explainability, we clearly see the proportion of CPU and memory in the latency prediction.

The variable selection is pivotal for interpretability as it indicates the importance of CPU and memory at each time step. By analyzing these influential features, we gain valuable insights into the underlying factors driving the forecasted latency compliance with SLA. The variable selection weights provide transparency into how the TFT model processes and weighs the CPU and memory. By associating specific weights with each latency prediction, we can understand the reasoning behind the decisions of the models and identify the key factors driving each forecasted latency outcome.

Similarly, multi-head attention is crucial for interpretability as it enables the model to focus on different parts of the input data and learn complex temporal dependencies. Multi-head attention allows TFT to compute attention weights for different CPU and memory features at various time steps. With these weights, we can equally interpret which of the CPU and memory is most relevant for making latency predictions at each time step. This provides insights into the relative importance of different features and helps us understand how the model processes and weighs input information when generating latency predictions. Additionally, multi-head attention enables the TFT to capture both local and global context when making predictions. By attending to different parts of the input sequence with attention heads, the model can integrate information from nearby and distant time steps to make more informed predictions. This allows us to interpret how the model incorporates both short-term and long-term dependencies in the data when generating the latency predictions. This detailed explanation of latency predictions enhances interpretability, improves analysis of SLA compliance, and provides actionable insights into the factors contributing to latency forecasts.

5.3.4 GNN for Spatio-Temporal Prediction

5.3.4.1 State-of-the-Art on GNN for Spatio-Temporal Prediction

As cloud and edge computing environments become increasingly dynamic, the integration of GNNs into workload prediction frameworks presents a transformative approach to managing these complex systems. GNNs, with

their capability to model intricate relationships and dependencies among data points represented as nodes and edges in a graph, are particularly suited for the volatile and interconnected nature of cloud-edge environments. These environments consist of numerous computation nodes where the workload distribution and network topology can change rapidly due to varying demand and resource availability.

The traditional models for workload forecasting in cloud and edge computing often struggle to adapt quickly to such changes, potentially leading to inefficiencies in resource allocation and increased operational costs. However, GNNs can leverage the inherent graph structure of these systems to capture the dynamic interactions between nodes—namely, the computation units across the cloud, edge, and far edge layers. This enables more accurate predictions of CPU usage and other critical resources by considering not only the state of individual nodes but also how they influence each other.

For example, in scenarios where certain nodes become congested, GNNs can help predict the ripple effects on adjacent nodes, allowing for proactive task offloading and resource reallocation. This is a step beyond static graph models, like those used in the GraphGRU model [39], which may not capture the temporal dynamics effectively. By employing models such as DySAT or EvolveGCN, which incorporate temporal changes and evolving graph topologies, the system can continuously update its predictions to reflect the current state of the network ([40], [41]). This dynamic adaptation ensures that resource prediction is not only real-time but also anticipatory, minimizing under-utilization and energy waste while enhancing the overall service quality and computational efficiency across the distributed computing infrastructure. Such capabilities make GNNs a pivotal technology in the ongoing development of smarter, more efficient cloud and edge computing strategies.

5.3.4.2 Workload Prediction for Volatile Nodes using Dynamic GNN

The primary objective of this study is to predict the future CPU usage of physical nodes within a distributed system. To achieve this, we leverage historical data on CPU usage from these nodes, as well as the CPU usage of various microservices that operate on them. These data points are not merely static but part of a dynamic interplay that unfolds over time across different nodes. To capture this dynamic, we are constructing a temporal graph that dynamically profiles each microservice.

5.3.4.3 Temporal Graph Construction

Given a set of physical nodes $N = (n_1, n_2, \ldots n_m)$ and a set of microservices $M = (m_1, m_2, \ldots m_k)$, each uniquely identified by an ID. Let U_n^t and $U_{m,n}^t$ denote the CPU usage of node n and the CPU usage of microservice instance m running on node at time t, respectively. Time is discretized into constant intervals, indexed by t, and $WINDOW$ is the number of time steps into the past included in the feature vector for both nodes and edges. The complete temporal graph G is defined as a series of temporal slices $\{G_t\}_{t=1}^{T}$, where each G_t corresponds to the state of the graph at time t. Each G_t is constructed as follows:

- Each node $n \in N$ is represented as a node in G_t.
- For each microservice m_i assigned to run on node n_x at time t, an edge is added from node n_y to n_x if m_i was running on n_y at $t-1$. Formally, the edge set E_t for graph G_t is defined as:
  $E_t = \{(n_y, n_x) \mid m_i \in M, ran_{t-1}(m_i, n_y), to\ n_x at\ t\}$
  where $ran_{t-1}(m, n)$ denotes that microservice m was running on node n at $t-1$.

Node and edge features at time step t are defined as:

- **Node Features:** For each node n at time t, the feature vector includes the CPU usage of the physical node n for the last $WINDOW$ timesteps: $\{U_n^{t-i}\}_{i=0}^{WINDOW-1}$.
- **Edge Features:** For each edge (n_y, n_x) representing the assignment of microservice m from n_y to n_x, the feature vector includes the CPU usage trace of the microservice instance m for the last $WINDOW$ timesteps while it was running on n_y: $\{U_{m,n_y}^{t-i}\}_{i=0}^{WINDOW-1}$.

5.3.4.4 Model Architecture

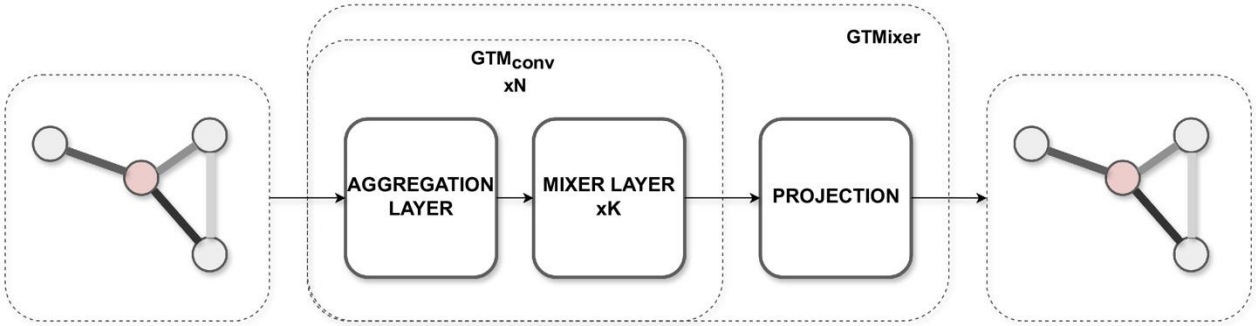

Figure 19: Model architecture

The GTMixer model is designed to leverage the temporal graph structure G_t using GNN principles. The model architecture (Figure 19) is composed of *N* Graph Temporal Convolution (GTConv) layers followed by a projection layer that outputs the predicted features as seen in Figure 20 below.

Aggregation Layer

Each GTConv layer operates on nodes u by aggregating spatial information from neighboring nodes v connected through edges e_{uv}. To perform this aggregation, a Single Head Attention mechanism is used as the first step. This mechanism dynamically weighs the influence of each neighbor v based on the edge attributes e_{uv} and the destination node features. The weighted features are then aggregated with a permutation invariant function such as max, mean, or sum, denoted with $\oplus$. This process is represented mathematically as:

$$h_u = \bigoplus_{v \in N_u} SingleHeadAttention(x_u, e_{vu}) \tag{16}$$

After aggregation, the spatial representation, which encapsulates the CPU consumption profiles from incoming microservices, is concatenated with the transformed representation of the source node. This node transformation is applied through a sequence that includes a linear layer, a ReLU activation, and dropout, with a residual connection to enhance the learning of subtle variations in the node features:

$$x'_u = x_u + \text{ReLU}\big(\text{Linear}(x_u)\big) \tag{17}$$

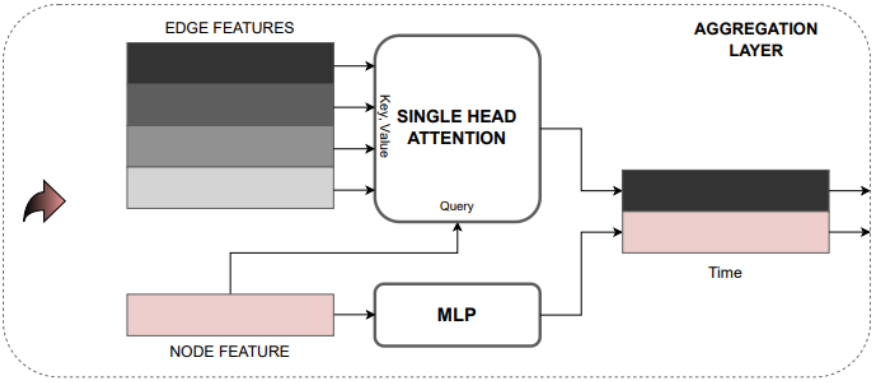

Figure 20: Aggregation layer

The concatenated vector is then passed through a series of MixerLayers, as described in the TSMixer architecture, which mixes features across both the node and its neighborhood to capture temporal dependencies effectively:

$$h'\_u = "\{MixerLayer\}(x'\_u||h\_n) \quad (18)$$

The Mixer Layer depicted in Figure 21 is made up of two modules: MLP_{time} and MLP_{feat}. MLP_{time} is like the source node transformation but with an additional normalization layer at the beginning. The MLP_{feat} module combines the source node features with the aggregated representation of the incoming microservices resource usage. It has a normalization layer, followed by two linear layers, ReLU activation, and dropout. This layered approach ensures that each node's feature vector represents its attributes and integrates information from its immediate graph neighborhood over time.

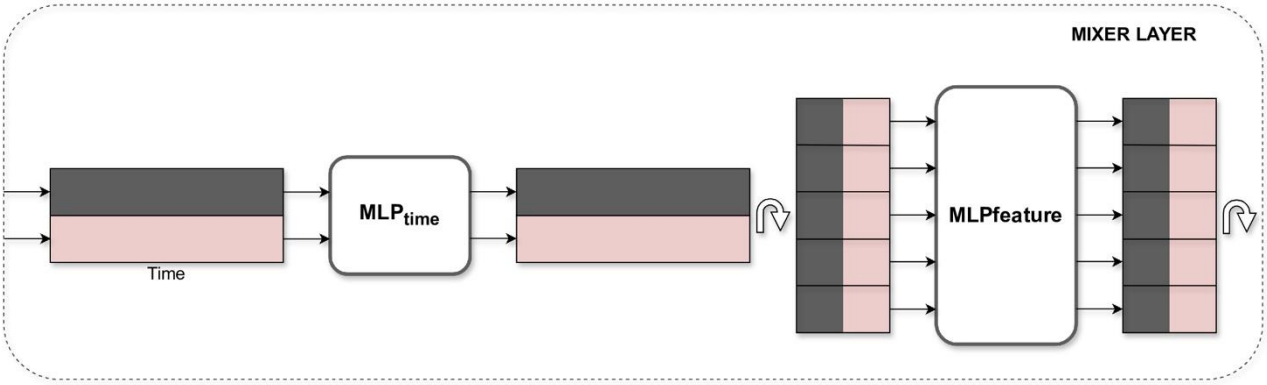

Figure 21: Mixer Layer

Temporal Projection

Finally, the output from the last GTConv layer is processed through a projection layer, which linearly transforms the high-dimensional feature vector to the desired output dimension, facilitating direct prediction of the CPU usage for each node. This model architecture aligns with the temporal graph G_t by explicitly considering both the static and dynamic aspects of nodes and edges, thereby enabling predictions that are deeply contextualized by the historical data embedded in the graph.

5.3.4.5 Performance Evaluation

5.3.4.5.1 Dataset

We analyzed a trace dataset from an Alibaba cluster, which included detailed runtime metrics of nearly 20,000 microservices. This dataset was collected from over 10,000 bare-metal nodes, documenting separate traces of CPU and memory usage over a 12-hour period in 2021. Upon analysis, we observed low variability in RAM usage across nodes, prompting us to exclude it from further experimental considerations.

To construct the temporal graph for our analysis, we utilized the node table, which included information on the CPU and RAM usage of physical nodes, and the MSResource table, which documented CPU and RAM usage traces of instances. Each instance was assigned a unique ID, a microservice ID, and the node ID of the physical node it was running on. In cases where a machine hosted multiple instances of the same microservice, we computed the mean CPU and RAM usage to establish the edge signals in the temporal graph. In each time step of our simulation, we randomly selected a percentage of nodes and erased their historical CPU utilization data to mimic the volatility inherent in edge environments. For these same nodes, we then recalculated the CPU utilization, U_n^t, by considering only the microservices incoming from the previous time step. This approach effectively simulates a scenario in which new nodes emerge and are assigned a subset of microservices to execute. For all experiments,

we consistently selected the same 300 nodes to ensure a fair comparison. We integrated the temporal information into our prediction model by defining a $WINDOW$ of 10 timesteps and a $HORIZON$ of 3 timesteps for the prediction algorithm. To evaluate GraphGRU we constructed the static relationships between nodes using the Dynamic Time Warping (DTW) algorithm.

5.3.4.5.2 Evaluation Metrics

To evaluate the performance of models predicting CPU usage in the context of the temporal graph G_t, we use both standard and dynamic evaluation metrics. Standard metrics include Mean Squared Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). The dynamic versions—Dynamic MSE (DMSE), Dynamic MAE (DMAE), and Dynamic MAPE (DMAPE)—specifically measure prediction errors for nodes that dynamically appear with no previous history.

Standard Metrics

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(U_n^{t_i} - \widehat{U_n^{t_i}}\right)^2 \tag{19}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|U_n^{t_i} - \widehat{U_n^{t_i}}\right| \tag{20}$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{U_n^{t_i} - \widehat{U_n^{t_i}}}{U_n^{t_i}}\right| \tag{21}$$

Dynamic Metrics

Dynamic versions of the metrics (DMSE, DMAE, DMAPE) follow the same formulas as their standard counterparts but are computed over n_{dynamic}, the number of observations for nodes that appeared dynamically without prior history.

5.3.4.5.3 Performance Evaluaiton

The results of our study are encapsulated in Table 3 which provides a comprehensive view of how each model performed across both standard and dynamic metrics. When comparing GTMixer with GraphGRU, the former achieved a substantial improvement, especially in standard metrics like MSE, MAE, and MAPE. These improvements are even more pronounced when considering dynamic metrics (DMSE, DMAE, DMAPE), demonstrating GTMixer's superior capability to adapt to nodes with no historical data.

Table 3: Results

Model	MSE	MAE	MAPE	DMSE	DMAE	DMAPE
GraphGRU	0.0377	0.1547	3.4180	0.0306	0.1435	3.2768
GTMixer (Ours)	0.0049	0.0436	0.0882	0.0154	0.0957	0.2202

6 Decision Enforcement with Reinforcement Learning

Cloud-native applications are inherently dynamic, experiencing varying levels of activity throughout their lifecycle [42]. With the cloud-edge ecosystem experiencing a significant influx of dynamic requests and continuous growth, the efficient management of resources to deal with diverse workloads becomes essential [43]. By leveraging real-time metrics on CPU and memory utilization, scheduling algorithms can take appropriate resource allocation decisions, preventing SLA breaches and resource wastage [44]. Moreover, resource management entails the efficient allocation of resources among various users or applications to maximize overall system utilization. Resource allocation policies need to take into account workload diversity, ensuring fair access to resources while preventing resource monopolization by individual users or applications [45]. In this deliverable, we propose a strategy based on Reinforcement Learning (RL) for allocating CPU and memory resources within a Multi-access Edge Computing (MEC) server that hosts heterogeneous services characterized by different SLA requirements.

6.1 Reinforcement Learning for Resource Allocation

RL is a model-free approach that does not require prior information on the system models' dynamics. It involves agents that learn optimal policies through gradually interacting with the environment [46]. Leveraging the abundance of data collected from heterogeneous networks [47], RL has become popular in recent years for developing methods that deal with the resource allocation problem. A preliminary study is presented in [48], combining Q-Learning and a heuristic approach for dynamically allocating resources on delay-sensitive services. In [49], Q-Learning is used to obtain the optimum level of maximum CPU usage in a function instance to trigger resource scaling decisions. Q-Learning-based approaches are also adopted in [50], [51] and [52] to determine resource scaling actions in order to maintain low application latency and failure rates. Authors in [53] use a Deep-Q-learning (DQN) based approach for resource allocation among different computation servers, while the works in [54] and [55] tackle the problem of computation offloading by breaking it into smaller subtasks involving RL algorithms. An actor-critic model has been implemented in [56] to optimize the allocation of radio and computing resources and proved to outperform existing benchmarks, such as Deep Deterministic Policy Gradient (DDPG) and DQN. Furthermore, a multi-agent distributed learning framework is proposed in [57] or making resource orchestration decisions, while federated learning is leveraged in [58] and [59] to optimally allocate communication and computing resources.

However, several key issues have not been addressed yet in the state-of-the-art literature. Computational load generated from heterogeneous services associated with the end users, edge computing or any other additional services is not considered together. In the present work, the following contributions are made to address the aforementioned limitations:

- An intelligent strategy is developed for dynamically allocating resources within a MEC server to heterogeneous services characterized by diverse demands and SLA requirements.
- The infrastructure's resource allocation is defined as a multidimensional problem with a known capacity, where each dimension represents a specific resource type, namely CPU and memory.
- To guide resource allocation decisions, we employ a Soft Actor-Critic (SAC) agent. Compared to a baseline random service allocator, the SAC agent demonstrates superior performance, achieving higher usage efficiency ratios, while adhering to the SLA requirements of the diverse services.

6.1.1.1 System Model and Problem Formulation

A MEC server S is considered with fixed computational C and memory M capacities that provides N different services to the end-users, as shown in Figure 22. The server considers the resource demands D_n of the services, which are represented as a resource tuple consisting of CPU and memory demands, and allocates the available

resources, which are also represented as a CPU-memory tuple (c_n, m_n). The services have heterogeneous SLA requirements, which as shown in Table 4 ([60], [61]), are expressed in terms of maximum permissible delay. The overall delay model is a combination of separate functions based on CPU and memory resources. For the computational delay, a simple model based on the CPU demand-allocation ratio is used:

$$l_c = \beta \frac{CPU_{demand}}{CPU_{alloc}} \tag{22}$$

where β is a weighting factor used for regulation. According to (22) when the CPU demand is higher than the allocated CPU, the delay incurred will be higher and vice versa.

Delay experienced due to memory utilization is modelled as a piecewise linear function, as shown in (23) [62]. There are two scenarios under consideration: (i) when the allocated memory meets or exceeds the demanded memory and (ii) when the allocated memory is lower than the demanded. In (23), $l_{surplus}$ and l_{scarce} are the minimum delays incurred under the two cases, while φ is the slope, which is derived based on reasonable delay values from the literature.

$$l_m = \begin{cases} l_{surplus} + \varphi l_{demand} & \text{if} \quad mem_{dem} \leq mem_{alloc} \\ l_{scarce} + \varphi l_{demand} & \text{if} \quad mem_{dem} > mem_{alloc} \end{cases} \tag{23}$$

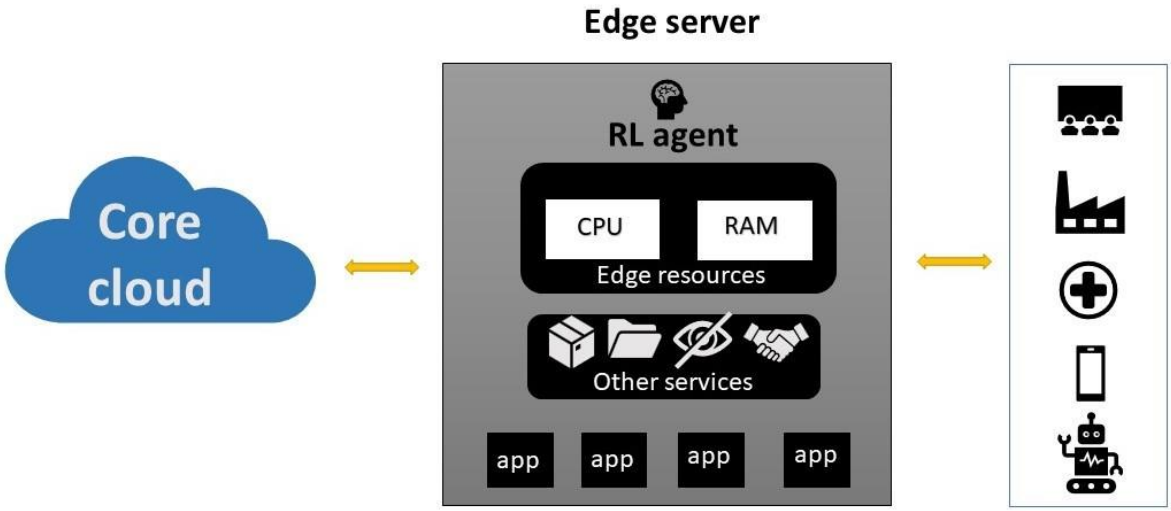

Figure 22: System model

Table 4: SLA requirements and Gn parameters

Type	Services	Max. Delay (s)	G_{CPU}	G_{mem}
I	Critical medical application I	0.02	22950	38050
II	Cyber-physical systems	0.01	32000	40000
III	Critical medical application II	0.1	34200	42800
IV	Video, imaging and audio application	0.00075	38450	23550

The resource allocation problem is formulated as a multidimensional problem with fixed capacity *R*. Each dimension represents the total CPU and memory capacity, denoted as *C* and *M*, respectively. The objective is to maximize the total utility by appropriately distributing the resources A_n to the *N* slices at different time windows as shown in (24):

$$U = max \sum_{n=1}^{N} G_n^{[CPU,mem]} A_n^{[CPU,mem]}(t) \quad (24)$$

where G_n represents the gain attached with each resource type for each service; the values of parameter G_n are determined through experimentation and are listed in Table 4. In addition, several constraints are considered to ensure strict surveillance of the allocation policy. Concretely, equation (25) limits the resource allocation based on the available capacity at the server. Equation (26) prevents SLA violation by maintaining the probability of allocated resources A_n being less than the demanded resources D_n, below a maximum acceptable threshold ε_n. Finally, equation (27) prevents over-provisioning by imposing a maximum resource allocation ratio m_n per service:

$$\sum_{n=1}^{N} A_n^{[CPU,mem]}(t) \leq R^{[CPU,mem]}(t) \quad (25)$$

$$\Pr\left(A_n^{[CPU,mem]}(t) < D_n^{[CPU,mem]}(t)\right) \leq \varepsilon_n \quad (26)$$

$$\frac{A_n^{[CPU,mem]}(t)}{D_n^{[CPU,mem]}(t)} \leq m_n \quad (27)$$

6.1.1.2 RL Formulation

The resource allocation problem is mapped into a Markov Decision Process (MDP), consisting of the state space, the action space and the reward function, as shown in Figure 23. At every time step, the state space s_t comprises of the server's maximum resource capacities *C* and *M*, the allocated resources A_n, the demands D_n, as well as the delays l_n at every network slice:

$$s_t = [C, M, D_1, \dots, D_N, A_1, \dots, A_N, l_1, \dots, l_N]. \quad (28)$$

The action space a_t comprises of the CPU and memory resources assigned to the slices by the RL agent:

$$a_t = [c_1, \dots, c_N, m_1, \dots, m_N] \quad (29)$$

The reward function in (30) guides the learning process by evaluating the actions taken at each time step:

$$r(t) = \sum_{n=1}^{N} U_n^{CPU} f(\delta_{CPU}) + U_n^{mem} f(\delta_{mem}) - P_n^{delay} \quad (30)$$

where U_n^{CPU} and U_n^{mem} are utility values associated with each resource type for each slice. The utility values are calculated by multiplying the allocations with the gains per resource per slice i.e., $U_n^{CPU} = G_n^{CPU} c_n(t)$ and $U_n^{mem} = G_n^{mem} m_n(t)$. The reward function also depends on the differences δ_{CPU} and δ_{mem} between the allocations made by the agent and the actual demand.

$$f(\delta_{resource}) = \begin{cases} 1 & \text{if } \delta_{resource} \geq 0 \\ 0 & \text{if } \delta_{resource} < 0 \end{cases} \quad (31)$$

In addition, P_n^{delay} in (32) penalizes the difference between the actual delay $l_n(t)$ and the maximum permissible delay l_n^{exp} for each slice type, as reported in Table 4.

$$P_n^{delay}(t) = \begin{cases} w_n\left(l_n(t) - l_{exp}\right) & \text{if } l_n > l_n^{exp} \\ 0 & \text{if } l_n \leq l_n^{exp} \end{cases} \quad (32)$$

The SAC algorithm is utilized for the resource allocation procedure, since it is appropriate for continuous state and action spaces. SAC finds a policy $\pi(a_t|s_t)$ i.e. a state-to-action mapping, that maximizes the sum of rewards with maximum entropy given by $J(\pi) = \sum_{t=0}^{T} E_{(s_t,a_t)}[r(s_t, a_t) + aH\pi(.|s_t)]$, in which α controls the stochasticity of the optimal policy and its importance with respect to the rewards. The algorithm finds this policy through an iteration process consisting of a policy evaluation step and a policy improvement step. In the evaluation step, a soft Q-function $Q(s_t, a_t) \leftarrow J(\pi) + \gamma E_{s_{t+1}}[V(s_{t+1})]$ is evaluated, where $V(s_t) = E_{a_t \sim \pi}[Q(s_t, a_t) - \log \pi(a_t|s_t)]$. By defining a modified Bellman backup operator [63] as T^{π}, a value sequence is obtained in which each value is given by $Q^{k+1} = T^{\pi} Q^k$. In the policy improvement step, π is updated towards the exponential of each new value of the soft Q-function. Every time the policy is updated, SAC guarantees that $Q^{\pi_{new}}(s_t, a_t) \geq Q^{\pi_{old}}(s_t, a_t)$. Updating π and $Q(s_t, a_t)$ in this way will yield an optimal policy. When considering continuous state and action spaces, Q , V and π are defined through deep neural networks (DNNs), which are optimized using stochastic gradient descent [64].

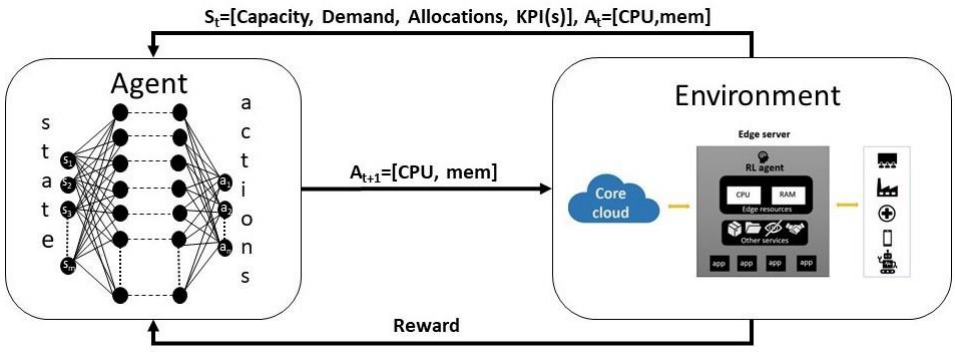

Figure 23: RL system

6.1.1.3 Performance evaluation

The proposed RL methodology is implemented in Python using Tensorflow 2.15 ([65], [66]). Table 5 specifies the parameters used for the SAC agent's training process, which have been appropriately tuned after experimentation within the considered infrastructure. The training dataset has been obtained from the Google cluster usage traces [67], with the CPU and memory demands of four different service types. A thorough preprocessing and scaling has occurred on the considered data to emulate real-world cases. The SAC agent's resource allocation decisions are compared with a random service allocator, which takes its decisions based on the gains of Table 4. Services with higher gains are favored under this approach. For each resource type and service, the two methods are compared in terms of resource usage efficiency and delay.

Table 5: SAC parameters

Learning rate	Episodes	Steps	Batch size	Optimizer	γ	α	τ
0.0001	20	1024	64	Adam	0.90	0.90	0.95

The results illustrated in Figure 24 and Figure 25 demonstrate the SAC agent's superior comprehension of service demands and available resources at the MEC server. As a result, it achieves a more favorable equilibrium between resource demand and allocations, leading to significantly higher CPU utilization ratios for service types I and II. It is noted that the proposed method attains similar outcomes for the remaining service types while adhering to all constraints outlined in equations (25), (26), (27). On the contrary, the random service orchestrator exhibits a notable over-allocation, resulting in resource wastage, which may affect the availability of resources on the server. Similar outcomes are presented in Figure 26 and Figure 27 for the memory allocations regarding service types II and IV, respectively. The SAC agent outperforms the random allocator, achieving much higher utilization rates. This also holds for the rest of the services, proving that the proposed RL-based method is capable of effectively managing multiple service types, maintaining high CPU and memory utilization rates.

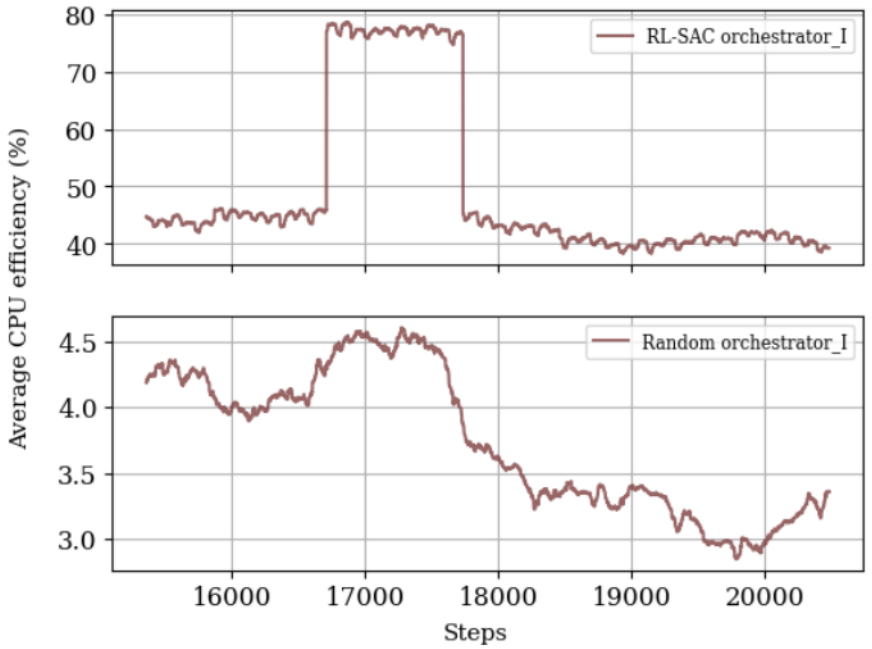

Figure 24: Average CPU efficiency for service type I

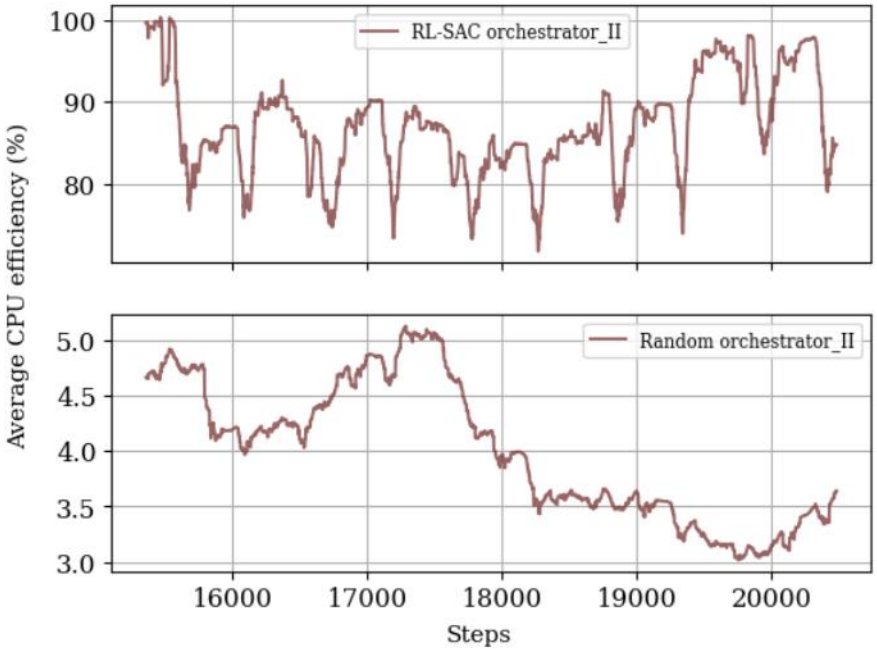

Figure 25: Average CPU efficiency for service type II

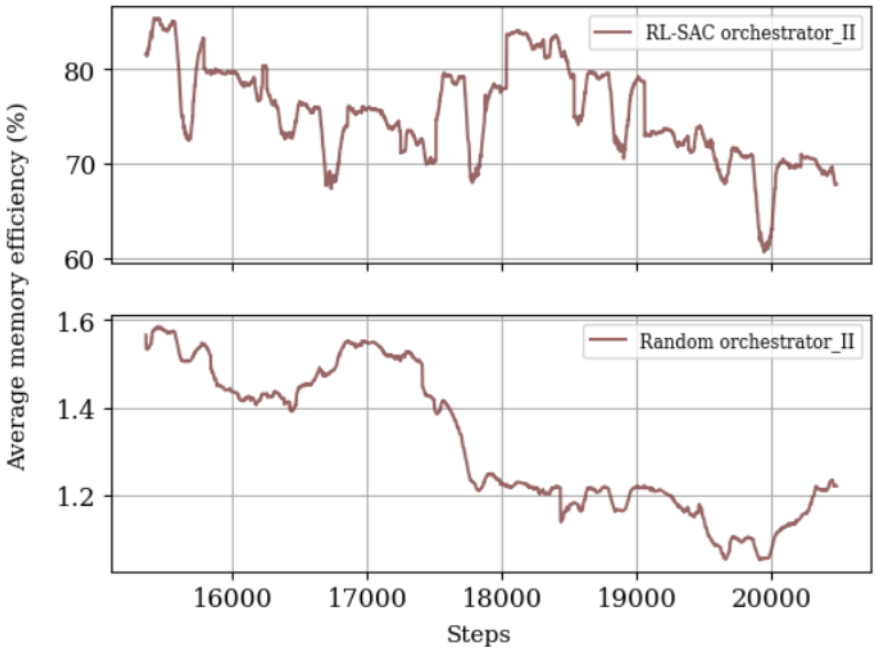

Figure 26: Average memory usage efficiency for service type II

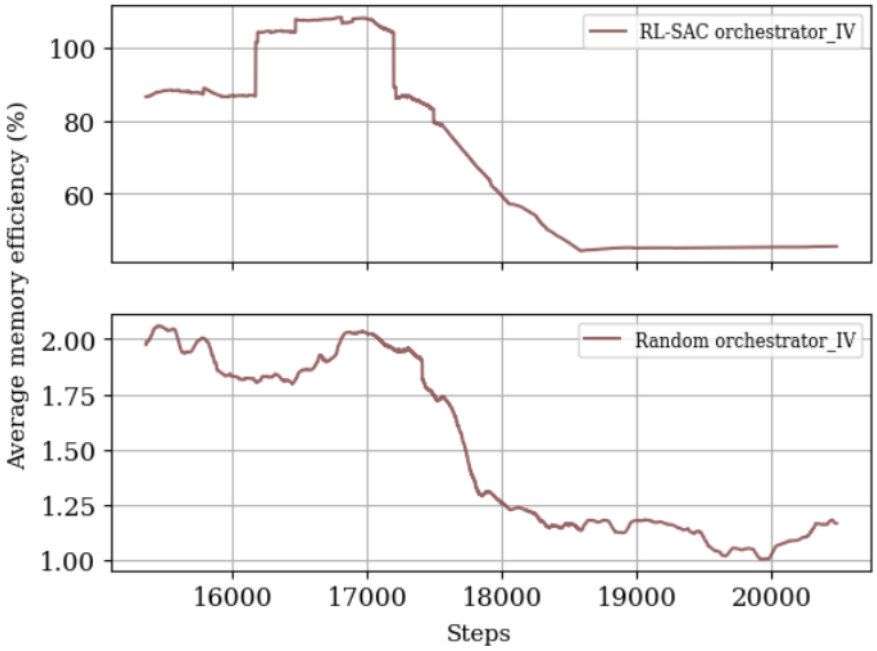

Figure 27: Average memory usage efficiency for service type IV

Moreover, SLA requirements for the services need to be met in order to avoid degradation in the smooth operation of the server. Figure 28 shows the maximum acceptable delay for service type III, as well as the average delays achieved by the two methods. Although both allocators successfully meet the set threshold, as previously discussed, this is achieved through a more efficient utilization of the available resources under the proposed SAC agent.

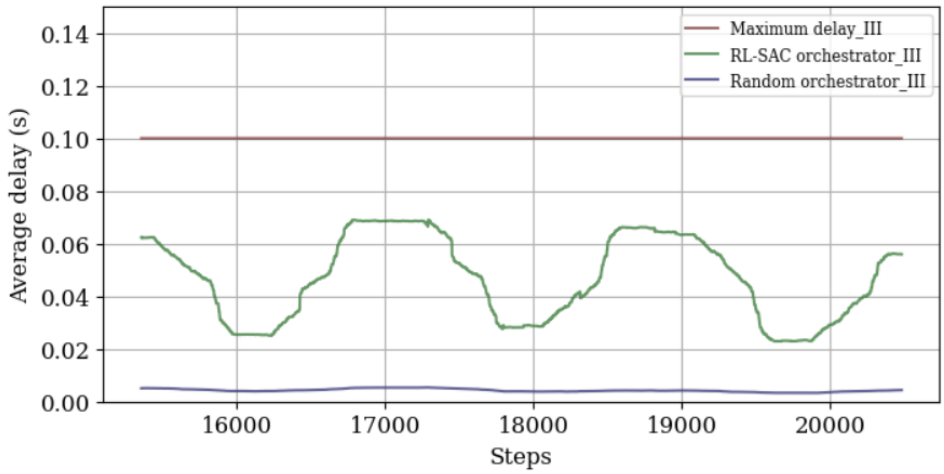

Figure 28: Average delay for service type III

6.1.2 Future Work on Reinforcement Learning for Energy-Aware Resource Allocation

Energy consumption in cloud-edge environments has shown rapid growth, leading to significant environmental and economic impacts [68]. Energy-aware resource allocation can address these challenges by optimizing resource utilization, minimizing energy consumption, while meeting performance requirements [69]. By leveraging advanced algorithms that dynamically adjust resources in response to varying user demands, dynamic workload patterns, heterogeneous resource characteristics, as well as latency and energy constraints, organizations can achieve significant cost savings, mitigate environmental impact, and ensure seamless user experiences [70].

Reinforcement learning holds immense potential for addressing the challenges of energy-aware resource allocation in the cloud-edge continuum. Authors in [71] have devised a RL-based strategy to offload tasks based on the service class and the energy they consume. A multi-agent RL algorithm is utilized in [72] to maximize the long-term energy efficiency within heterogeneous networks. A resource allocation policy with power constraints for IoT-based networks is developed in [73], while the concept of reward-oriented task offloading under limited power resources is examined in [74].

However, there is a lack of focus in some important areas in the state-of-the-art literature. Many studies overlook the sensitivity of tasks or services, neglecting their impact on energy consumption at the network edge. For instance, a delay-sensitive task may demand more energy compared to a compute-sensitive task, or vice versa. Understanding the specific resources utilized at the network edge is an aspect that has not been thoroughly explored. Furthermore, the findings presented in most state-of-the-art research heavily rely on a single Key Performance Indicator (KPI), which might not accurately represent real-world scenarios.

Taking into consideration the aforementioned limitations, in our future work, we are planning to expand the model presented in the previous section developing an energy-efficient resource allocation strategy based on RL. Currently, key aspects of the proposed model, such as the representation of the state and action spaces, as well as the reward design, are under development. Our target is to consider an appropriate state space that captures relevant information about the system, including workload characteristics, resource availability, as well as energy consumption and resource utilization metrics. Moreover, the action space should include a set of decisions that the RL agent takes to optimally allocate resources, such as vertical and horizontal scaling. Finally, special focus will be given on formulating a reward function that incentivizes energy efficiency while penalizing deviations from performance objectives, such as delay or throughput requirements.

7 Network Programmability

AC³ specifically deals with federated cloud-edge infrastructures, and as such, the network infrastructure that supports it must also be designed and managed with federation in mind. When dealing with such types of infrastructure, there are several core factors that need to be addressed, specifically, transience and heterogeneity:

- Federated infrastructures are by their nature transient, meaning that the environment can, and will, change, dictating that we cannot make assumptions about fixed infrastructures being in place. Therefore, AC³ must be capable of dynamically creating and configuring networks, as well as adapting the network to the underlying infrastructure or application deployments.
- Federated infrastructures are also, by their nature, heterogeneous, meaning that the environment will be made up of a broad range of different technologies carrying out the same or similar functions. This is no different in respect to networking, especially in the world of virtualized networks, where more control is given to engineers (developers/dev-ops) to build and create their own.

Therefore, the AC³ architecture must consider that individual infrastructure providers may utilize a number of differing technologies and approaches in providing network connectivity, while at the same time having the flexibility to adapt to changing infrastructure. Within this task, we address this through Network Programmability, using virtualized networks that can be created, modified, and removed on demand. At the same time, we explore how differing programmable network technologies can be used and managed within the AC³ architecture.

Specifically, we investigate using both SD-WAN and Kubernetes-based networking as two common approaches to providing programmable networking in the context of containerized applications. Both SD-WAN and Kubernetes effectively address the transiency and heterogeneity issues facing federated infrastructures. SD-WAN optimizes network paths, providing consistent performance and resilience in transient environments, while Kubernetes mitigates heterogeneity by creating a unified orchestration layer that allows for consistent management of resources, including networks. The combination of both enhances network and application performance and provides uniform interactions across disparate environments.

7.1 State-of-the Art on Network Programmability

In the context of the AC³ project, we are focusing on two particular approaches to Network Programmability. Specifically SD-WAN and Kubernetes-based network orchestration.

7.1.1 Software Defined Networking

The academic research on SD-WAN, in general, appears to be at an early stage, with very few research works done on interconnecting cloud/edge/far edge using SD-WAN, particularly in cloud edge continuum. In addition, besides considering works dedicated to only cloud interconnection, in this section, we have included works that focus on enterprise networking and those that suppose a complete control of the underlay network, though, in AC³ context, we are mainly concerned with interconnecting nodes, without having control over the underlying networks.

[75] addresses the challenge of resiliency in an SD-WAN environment basically composed of two CPEs and dual WAN links. They focus on traffic engineering for dynamic management and prioritization of network flow. Leveraging Floodlight Controller, their architecture is composed of a Java-based SDN controller, and custom software modules on top of it for traffic engineering. Authors in [76], propose an SDN-based network architecture to improve cloud resiliency by interconnecting data centers through an overlay network managed by a centralized controller. The overlay network represents different egress nodes, each acting as a gateway for

data centers positioned behind them. These egress nodes are connected to the Internet through different Internet Service Providers (ISP). Their architecture utilizes Software Defined Networking (SDN) and Segment Routing (SR) to dynamically adapt routing in response to network failures. Authors in [77] implement a testbed supporting SD-WAN to connect two data centers. The goal is to guarantee the predefined QoS and traffic prioritization. The controller was able to efficiently manage 300 VoIP calls, using a maximum of 16% CPU load.

B4 ([78][79][79], [79]) represents the software-defined inter-datacenter WAN deployed by Google. SD-WAN B4 connects data centers in different locations using a two-tier hierarchical control framework. At the lower layer, each data center site contains a network controller and hosts local control applications managing site-specific traffic. Meanwhile, at the top layer, a logically centralized traffic engineering server is implemented. This server enforces high-level traffic engineering policies aimed primarily at optimizing bandwidth allocation between competing applications across different data center sites [80]. Customized switches were designed to fit B4, taking into account Google's inter-datacenter WAN and traffic characteristics. In B4, the network controller dynamically reallocates bandwidth to meet evolving application needs while also offering dynamic rerouting in the event of a link or switch failure. [81] introduces SWAN, a Software-Driven WAN system designed to improve the utilization of inter-datacenter networks. By centrally controlling service traffic and reconfiguring the network's data plane to align with current traffic demands, SWAN avoids transient congestion often caused by uncoordinated updates in traditional networks. They showed that leaving 10% free link capacity allows for quick congestion-free updates.

[82] presents an SD-WAN demo-test implementation leveraging open-source tools, focusing on improving enterprise network services. Their implementation connects two branch offices to a headquarters, using OpenDaylight [83] for SDN control, OpenvSwitch [84] for virtual switches, and monitoring services for dynamic path selection. VyOS [85] was used for WAN virtual router emulation.

There are other open-source SD-WAN solutions like FlexiWAN [86] and EveryWAN [87]. In [86] the controller is implemented based on Free Range Routing [88], and the edge device or route infrastructure using FD.io VPP [89], and it envisages a more classic approach to SD-WAN with the control functionalities still running at the edge in the virtual routers. EveryWAN, on the other hand, uses a hybrid IP/SDN approach where a local control logic based on distributed IP routing coexists with a programmable IP forwarding engine controller by a SD-WAN controller [87]. The Universal Customer Premises Equipment (uCPE) was implemented based on Linux networking leveraging the pyroute2 [90] netlink library.

7.1.2 Kubernetes-based Network Orchestration

Container Network Interfaces (CNI) form the foundation for networking within Kubernetes, providing a container with a network interface, IP addresses, subnets, and routing rules [67]. The CNI abstraction allows Kubernetes to remain agnostic to the underlying network while ensuring that pods can communicate across nodes in the cluster. There are various CNI implementations (plugins) which deliver a range of networking capabilities to the cluster.

- Calico places an emphasis on strong network security through advanced policy enforcement, supports both overlay and non-overlay networks, and is capable of running in large scale and multi-cluster production environments [68].
- Flannel creates a mesh of layer 3 network subnets which are assigned to each Kubernetes node and is particularly effective in environments where simplicity and ease of deployment are prioritized. It does not manage network policies but can be used in conjunction with those which do and can be configured to provide cross-cluster communication [69].
- Cilium provides the capability of recognizing, interpreting, and managing traffic based on application level APIs which allows the network to understand and interact with the packets being transferred [70].

Cilium handles protocols such as HTTP, gRPC, and Kafka at the network layer, enabling security measures and controls specific to each protocol. CiliumMesh could provide networking capabilities which enable the connection of multiple clusters but relies on the Cilium CNI which must be adopted in each cluster.

While CNIs can be extended to support multi-cluster connectivity, their primary focus is intra-cluster networking. This multi-cluster support is an additional capability rather than their main function. In contrast, CNI agnostic tools like Skupper and Submariner are specifically designed for multi-cluster connectivity. They provide a more straightforward and efficient link between clusters, with fewer prerequisites and less overhead.

Kubernetes provides its own set of network resources such as services, NodePorts, and Load Balancers, that sit on top of the CNI, leveraging it's capabilities to provide varied network features. For instance, ingress resources are used to provide HTTP and HTTPS-based access to applications from outside a cluster.

However many of these features are highly oriented towards intra-cluster communication. Using these resources to address inter-cluster communication can require significant configuration, can lead to compromised security posture, or are often limited to HTTP only.

7.1.2.1 CNI-Agnostic Solutions

Skupper is a multi-cluster service interconnection tool for Kubernetes. In contrast to broader solutions like Submariner and Cilium, Skupper connects specific namespaces rather than entire clusters through a Virtual Application Network (VAN). It leverages virtual addressing to mirror services across clusters, simplifying integration and improving security. The benefits of this include enhanced service discovery and inter-cluster communication, utilizing mutual TLS for secure connections, and isolating traffic to prevent network attacks. This targeted approach reduces the complexity typically associated with multi-cluster configurations [71].

Submariner provides a simple and direct networking solution for Pods and Services across different Kubernetes clusters hosted either on-premises or in the cloud. As an open-source tool that is agnostic to network plugins (CNI), Submariner uses a centralised broker architecture to manage cluster configurations and connectivity, and straightforward service discovery within the same cluster [72]. Key advantages include an easy setup with a broker to handle cluster credentials and efficient routing management. Traffic between clusters passes through designated gateway nodes which are selected via leadership elections. This ensures a secure and structured data flow and is optimal for environments requiring inter-cluster communication without the complexities of multi-cluster services.

Istio provides a set of tools to manage microservices across clusters without being dependent on a specific CNI as with CiliumMesh and Cilium. Its advantages include advanced traffic management, automatic mutual TLS for secure communication, scalable policy enforcement, and observability through detailed telemetry [73].

Cloud-Native Wide Area Networks (CN-WAN) are designed to manage service connectivity across clusters and service providers, independent of CNI's. CN-WAN can provide optimized network traffic routing based on service metadata, reduced latency through intelligent path selection, and dynamic network adaptation to support enterprises in efficiently utilizing WAN resources to enable increased performance in a multi-cloud environment [74].

Linkerd is a lightweight service mesh that is not dependent on a specific underlying CNI and focuses on enhancing the security, speed, and reliability of service communications. Key advantages include minimal configuration, automatic proxy injection, real-time failure detection and response, and extensive observability functionality.

7.2 Proposed Hybrid Architecture for Multi-layer Network Programmability

As discussed in Section 7.1, our Network Programmability solution must be able to cater for both transience and, in particular, heterogeneity of infrastructure. The broader AC^3 architecture addresses this through the

Adaptation and Federation layer, which acts as an adapter to bind to the specific APIs of the Local Management Systems (LMS). The Network Programmability task of AC³ attempts to take this further by making no assumptions on the specific network technology in use. That is, in the same way that the AC³ architecture presumes the existence of multiple LMS for the management of compute resources, we also assume that there may be multiple LMS for the creation and management of network resources. The LMS for the network can be based on multiple technologies, here we focus on 2 broad types of networking; namely SDN, specifically SD-WAN, and CBN (Kubernetes). In Figure 29, we show an example of the 2 technologies working in tandem to support the networking needs of AC³. Here we see the LCM, which based on its acquired knowledge of the application and resource utilization profiles, orchestrates the placement and deployment of the microservices, via the Decision Enforcement and Adaptation Layer. As part of this process it provides the relevant deployment configuration information of the microservices to the appropriate Network LMS: either the SD-Controller for managing SD-WAN based data-centre networks; OR the AC³ Network Operator for managing Kubernetes-based data centre networks. The selected LMS then attempts to configure the network in order to implement the required connectivity.

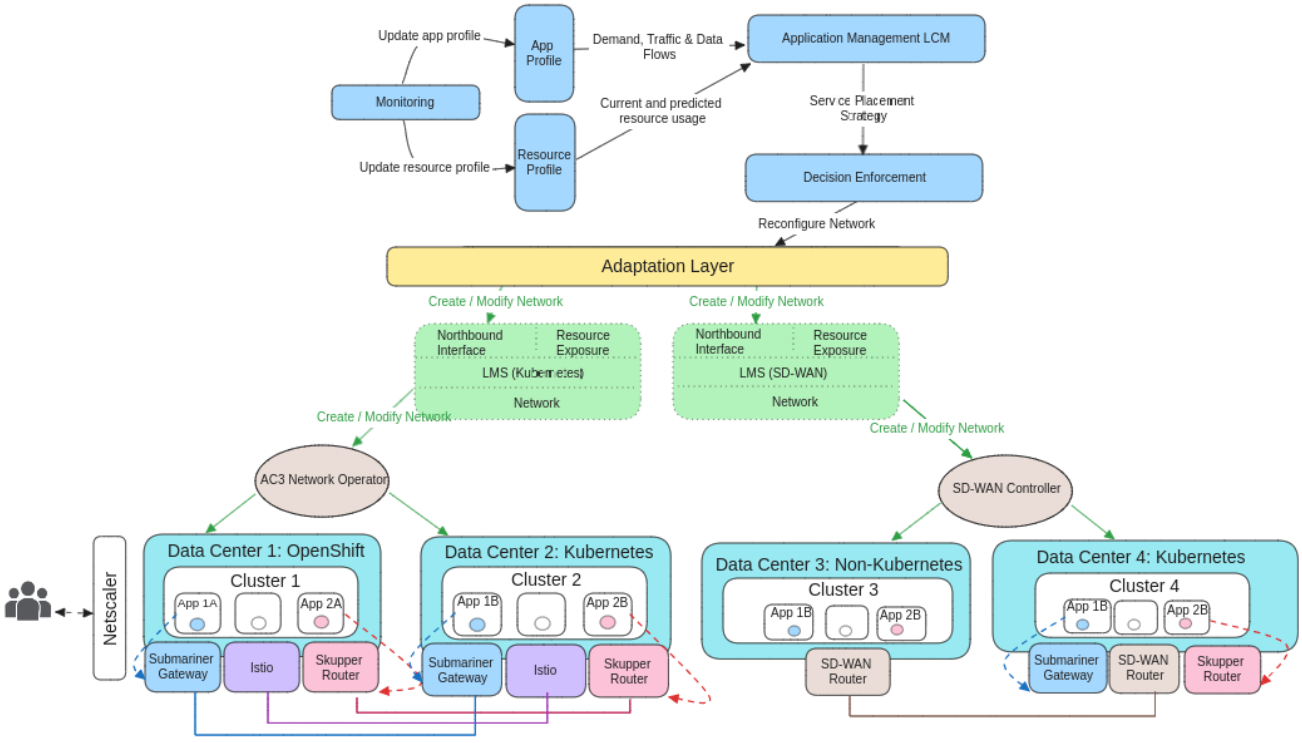

Figure 29: Software Defined Networking and Container-Based Networking working together

In the context of AC³, we can classify Network Programmability functions into 2 broad categories:

- Proactive Network Configuration: The ability to configure networking on demand to support specific application deployments or infrastructure configurations.
- Reactive Network Configuration: The ability to reconfigure the network in line with detected QoS or QoE issues.

7.2.1 SD-WAN

7.2.1.1 SD-WAN Overall Architecture

The proposed architecture abstract and aggregate the different networking resources (multiple heterogeneous WAN paths) interconnecting the CECC nodes to the CECCM, and to introduce flexibility, agility and programmability in managing deployed microservice's traffic flow based on the defined SLA/QoS requirements, and the network conditions. The overall architecture is composed of three planes: i) management, ii) control, and iii) data plane, as depicted in Figure 30.

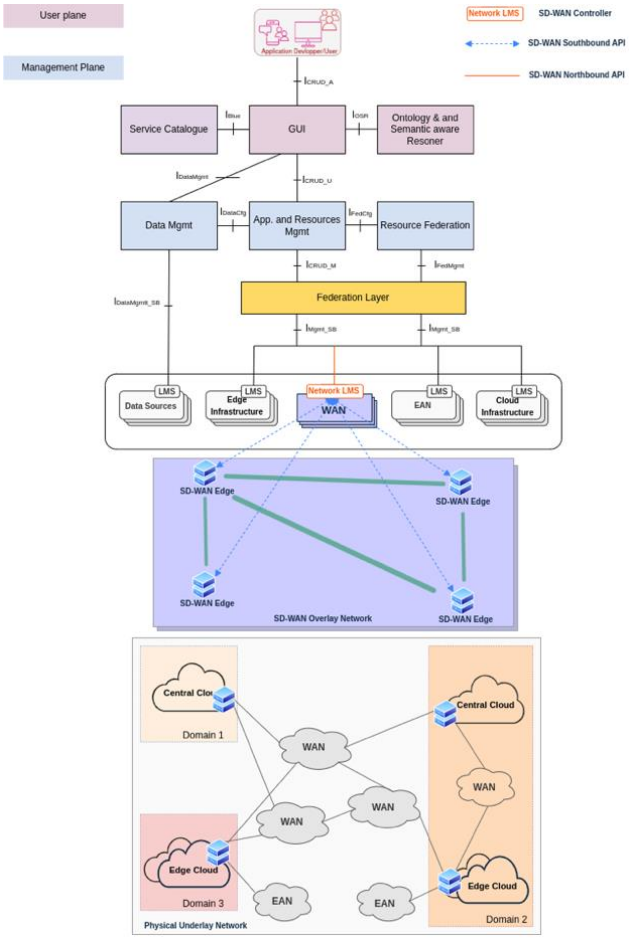

Figure 30: CECC SD-WAN Overall Architecture

Management Plane

As the overall SD-WAN architecture illustrates, the SD-WAN management plan is an integral part of the CECCM. The management of WAN interconnecting CECC nodes belonging to different infrastructure providers is part of the resource management in the CECCM management plane. The federation layer, which acts as an abstraction layer between the management components and the various local management systems, can also federate several SD-WAN controllers (WAN LMS) in the case of a distributed SD-WAN control plane. In addition to computing infrastructure like cloud and edge LMS exposing APIs to the CECCM, the SD-WAN, acting as an LMS for the networking resources part, exposes network management functionalities to the CECCM using its NBI. It bridges the gap between the abstracted networking capabilities provided by the SD-WAN controller and the

operational demands of the App and Resources Mgmt component of the CECCM.

As the CECCM orchestrates the placement, deployment, and LCM of microservices, it is able to provide relevant deployment configuration information to the SD-WAN controller identifying microservice traffic. This enables the controller to configure SD-WAN edge devices with matching rules (like protocol, port number, source/destination IP addresses, etc.) to identify and classify application traffic. These classes of traffic are grouped based on QoS requirements, and different microservices can be classed into one class according to their similar QoS requirements. These classes are defined at the CECCM level and passed to the controller in the form of requests to be then pushed by the controller as configurations to the edge devices as matching rules, which will be used for traffic identification and classification.

For instance, consider a microservice experiencing SLA degradation, such as exceeding the maximum acceptable latency threshold while its traffic flows through overlay network “ON-1”, the CECCM, assessing both the overall QoS requirements specified by the application developer and the current conditions of the network leveraging a monitoring module, will request rerouting the microservice's traffic to a different overlay network “ON-2” that satisfies the QoS requirements.

Subsequently, the controller implements policy adjustments to redirect the traffic through the tunnels of the overlay network “ON-2”. These traffic updates could be static, specified by the application developer or the CECCM user, or dynamic using an integrated AI/ML-based module such as RL.

In case more than one overlay link is satisfying the QoS requirements of a specific traffic class, the CECCM can load balance traffic over overlay links that satisfies the QoS requirements of this class (e.g. per flow load balancing). Another CECCM's management plane functionality is the deployment of edge devices, Virtual Network Functions (VNFs), after adding a new CECC node.

Control Plane

Decoupled from the data plane, the control plane is fundamentally responsible for enforcing the control and configurations of the edge devices. Through the NBI, the SD-WAN controller (Network LMS) receives network update requests from the CECCM, and then translates these requests to a set of commands or configurations to perform at the edge devices leveraging the SBI.

In addition to central management of the edge devices's networking aspects, from interface configurations and overlay tunnels to routing tables and advanced policy-based routing decisions, the controller also handles the monitoring agents on edge devices that monitor overlay tunnel specific network metrics (e.g., latency and packet loss).

This global view of the network, in terms of different overlay topologies, and real-time network performance, exposed to the CECCM can be leveraged for dynamic traffic management, and optimizing the microservice SLA (network QoS requirements) fulfillment.

Data Plane

In a typical SD-WAN solution, for example, in enterprise networks, the data plane connecting different enterprise branch sites and headquarters is established by creating overlay links over both private and public IP/WAN infrastructures. However, in the CECC context the private IP/WAN connectivity is not always guaranteed, and cloud/edge resources could be provided by different providers that do not share dedicated private IP/WAN links.

In order to connect the geographically separated CECC nodes and abstract the heterogeneous WAN connections (broadband LTE/5G, Internet, MPLS, ect.) logical overlay links are created over the existing physical underlay WAN infrastructures. The SD-WAN overlay network consists of the edge devices (also known as CPE/vCPE in enterprise networking) and the set of logical tunnels created between these edge devices. Each edge device is deployed at the border of each CECC node as a gateway between the CECC node’s (cloud or edge) local network

and the different WAN; hence, representing an endpoint for overlay tunnels.

These tunnels are created using different technologies such as VXLAN [91] or GRE [92] over IPsec to encapsulate traffic between CECC nodes, thus providing secure communication paths and allowing flexibility in forwarding behavior. The configurations of edge devices for instantiating overlay tunnels, defining policies, recognizing and classifying traffic for application-driven routing and routing decisions are centrally managed at the control plane and pushed by the SD-WAN controller to the edge devices leveraging its Southbound API.

In order to be able to dynamically route traffic of different services deployed in different CECC infrastructures, edge devices, in addition to overlay packet encapsulation, need to be able to identify or recognize microservices traffic based on defined policies and matching rules in order to route it through a specific overlay based on defined QoS requirements or SLA agreements and network conditions, enabling an application-driven routing which is detailed in the following section.

7.2.1.2 Architecture Instantiation

This section presents an instance implementation of the proposed architecture. The architecture depicted in Figure 31 has been implemented based on the open source SD-WAN EveryWAN tool. We started by leveraging the implementation of both the edge device and the controller of EveryWAN, and the adaptation agent of the CECCM for interacting with the controller NBI (Annexe A).

In EveryWAN, the SD-WAN edge device was implemented as VNF, and can be deployed on any server providing computing, storage and network interfaces. Designed with a hybrid IP/SDN approach, this SD-WAN edge device combines a Programmable IP Forwarding Engine (P-IPFE), an IP routing daemon (IPRE) and a Southbound API also implemented using gRPC. Besides programming the forwarding entries, the controller can override the routing decisions taken by the IPRE. The IP Forwarding Engine was implemented using Linux networking, with the pyroute2 python library facilitating interactions with the Linux kernel netlink interface.

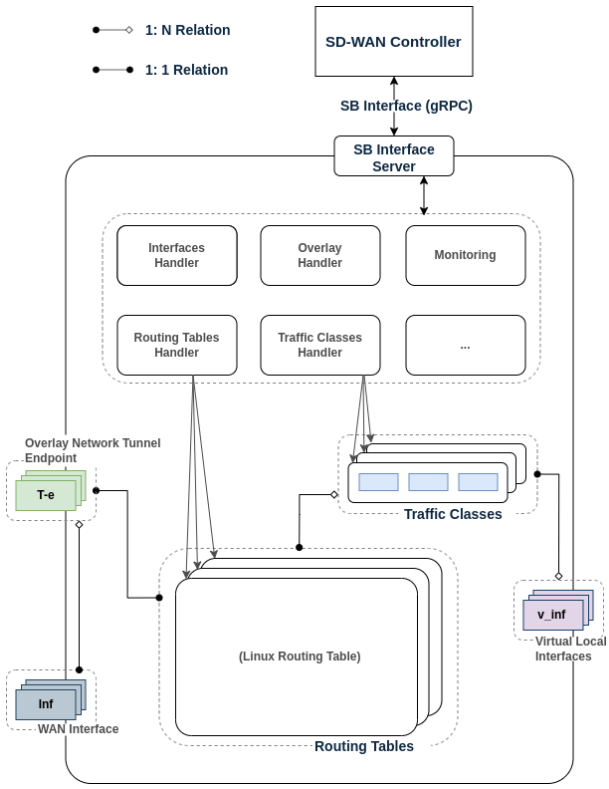

Figure 31: Application aware routing

For traffic redirection of a specific application over a specific overlay network, It exists an E2E network slice in their SD-WAN solution as a combination of LAN/VLAN interfaces and an overlay network. For isolation between applications traffic in different overlay networks, a VRF lite approach was adopted mapping, for each overlay network, one LAN/VLAN interface with one VRF and with one physical WAN interface. However, besides the limitation of supporting only one underlay WAN interface in EveryWAN implementation, the E2E Slice definition (vLAN - overlay network - vLAN) using VRF lite presents another limitation in our case. As an interface cannot be slave to multiple VRF at the same time, the association of several overlay networks created on different underlay WAN with the same LAN interface was not possible with VRF lite, knowing that virtual WAN interfaces and bridging would raise complications.

To address these challenges and adapt to the CECC context, we reimplemented certain modules like the overlay manager in the SD-WAN controller, and the different handlers in the SD-WAN edge. Additionally, we added other needed modules like application-driven routing and monitoring.

Firstly, to support hybrid WAN connections, and enable overlay network over multiple underlay links without losing the Layer 3 isolation, we opted for utilizing Linux routing tables as instances of P-IPFE instead of VRF.

Secondly, for the application-driven routing, as depicted in Figure 31 each Linux routing table of one overlay network is mapped to one or a group of traffic classes instead of a one LAN/vLAN interface mapped to one VRF instance. Each overlay network has a dedicated Linux routing table, thus enabling logical isolation from other overlay networks and allowing microservices and applications from different overlays to have an isolated routing behavior. Thus, we are not limited to the local interface to separate traffic, and each traffic class can have multiple ingress interfaces (LAN/VLAN interfaces).

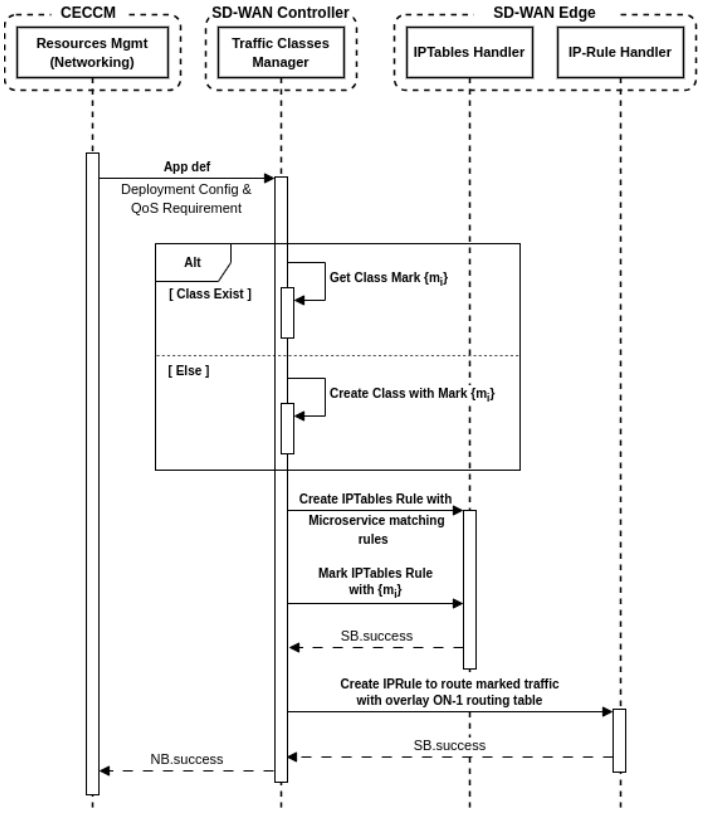

Figure 32: Service traffic identifier creation process

Figure 32 illustrates the process of defining an application for traffic classification and routing over a specific

overlay network (ON-1). As mentioned in above sections, the CECCM has a global view of applications deployment, and information like deployment configurations and matching rules (e.g. protocol, port number, source/destination ip addresses, etc.) can be provided as a JSON file to the CECCM network manager. This information is used by the network manager to identify and classify service traffic, which then requests the SD-WAN controller to the corresponding rules to the traffic classes. At the edge level, we used Linux IPTables tool for traffic classification and marking and IPRule for redirecting the traffic marked with the traffic class mark over the overlay routing table.

In addition, one of the main functionalities of the CECCM is dynamic rerouting in response to any experienced SLA degradation, specifically microservice QoS requirement violation. For this, we have added overlay monitoring to provide network metrics to the CECCM resource monitoring module, that can be used by an AI-based traffic engineering module, for example, an RL algorithm for dynamic routing. In our initial implementation, we adopted a straightforward active monitoring approach, in which network measurements including latency and packet loss are periodically transmitted to the controller.

7.2.1.3 Performance Evaluation

In this section, we present the results of our evaluation tests, which include edge-level overlay creation time, E2E overlay creation time, memory and CPU consumption of the SD-WAN edge device (VRF vs. Linux routing table). The performance evaluations were conducted using a test bed comprising 2 edge devices and one SD-WAN controller, each implemented as virtual machines (VMs). The edge devices were connected using two different laboratory networks and the properties of a WAN were emulated using NetEm. The SD-WAN edge VMs were allocated 2GB of RAM and 1 CPU, while the controller VM was allocated 4GB of RAM and 2 CPUs. The host machine on which the edge device virtual machine was provisioned was equipped with an Intel® Core™ i7-1365U processor with 10 cores, and 3.9GHz clock speed, and 32GB LPDDR5-6400MHz RAM.

7.2.1.3.1 Overlay Creation Time

Figure 33 shows a comparison of the time required to create SD-WAN overlay tunnels in a single SD-WAN edge between our approach using Linux Routing Table (RT) approach and the VRF approach. Initially, with overlay numbers under 100, both the VRF and RT approaches show a nearly constant time, followed by a linear increase after 100 overlay tunnels. Overall, both approaches had similar overlay creation time with minor deviations.

Figure 34 represents the E2E time taken to create overlays from the MMO request moment to the overlay tunnels establishment. These results closely resemble those in Figure 33 despite the slight delay due to the laboratory network latency between the MMO/SD-WAN controller and the SD-WAN edges.

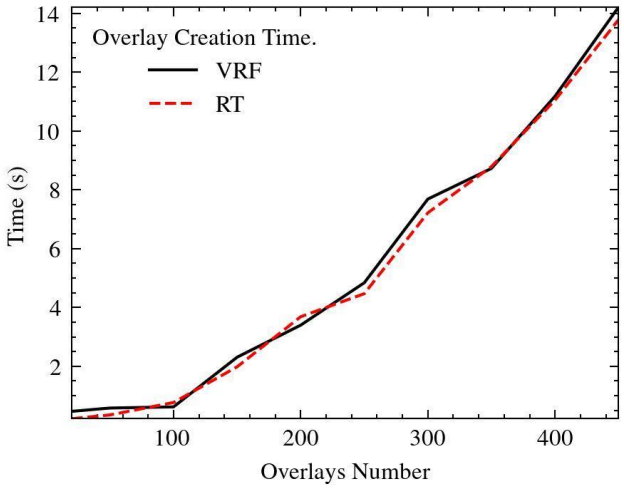

Figure 33: Overlay creation time

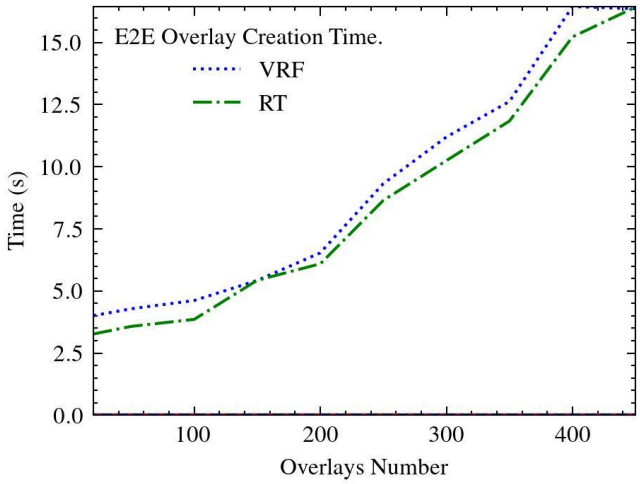

Figure 34: E2E overlay creation time

SD-WAN Edge CPU Usage

The evaluation presented in Figure 35 considers the CPU usage of the SD-WAN edge Virtual Machine in terms of the overlay tunnels number created at each time in both the VRF and the RT approach. The findings show a linear rise in CPU usage as the number of overlay tunnels increases for both the VRF and the RT approach. The overall performance of both approaches is similar, although the RT approach tends to have slightly lower CPU usage.

SD-WAN Edge RAM Usage

Figure 36 illustrates memory usage for creating overlay tunnels, comparing between our approach using the Linux RT and the VRF approach. The findings illustrated in Figure 36 show a linear, slow-rate increase in RAM usage as the number of the overlay tunnels increase. Comparatively, the RT approach demonstrates memory efficiency, consuming around 10% less memory than the VRF approach. This difference can be attributed to the additional overhead associated with running multiple VRF instances.

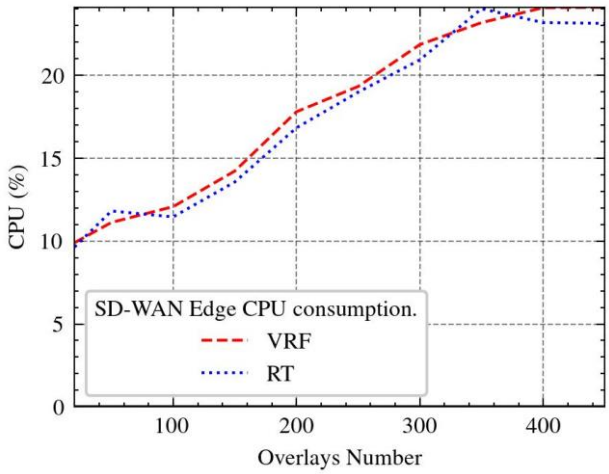

Figure 35: SD-WAN edge CPU consumption

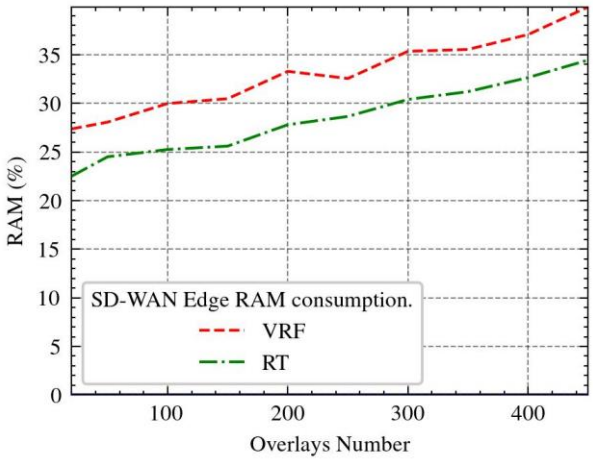

Figure 36: SD-WAN edge RAM consumption

7.2.2 Kubernetes-based Network Orchestration

In terms of containerisation, undoubtedly one platform stands out as being the leader, and can almost be considered the de-facto standard for container execution, management and orchestration. Kubernetes boasts a host of features that make it the platform of choice for organisations of all sizes across industry. However, some of the most powerful features of Kubernetes, and the ones that make it so suitable for building our Network Programmability solution on, are [75]:

- **Declarative Configuration:** is a foundational feature of platforms such as Kubernetes. The desired state of the network is detailed in a configuration file rather than specifying the steps required to achieve that state. This streamlines network management, reduces errors, and drastically improves reproducibility across different environments.
- **Extensibility**: allowing users to define custom resources that can be created and managed in the exact same way as any first-class platform resource.
- **Common API**-based management leverages standardised APIs to orchestrate and monitor network resources in Kubernetes. A unified interface allows developers to interact with services and plugins

without having to alter the underlying network. This provides enhanced flexibility, scalability, and adaptability with regards to the networking framework.

In this work we leverage these core platform features to deliver the AC³ Network Operator. The aim of this component is to provide a novel management layer for multi-cluster network configuration and management. Specifically, this component aims to:

- Implement the AC³ LMS for Kubernetes, enabling AC³ to manage and control K8s based networks
- Enable automated creation and configuration of multiple network overlays of varying types within Kubernetes
- Dynamically adapt the network configuration based on monitoring data in order to improve performance
- Be extensible, in order to support the integration of additional Kubernetes network technologies, supporting multiple overlays of varying types

7.2.2.1 AC³ Network Management Operator

As shown in Figure 37, the AC³ network operator implements the network LMS interface to the broader AC³ architecture, allowing AC³ to manage and control the network configuration within a given Kubernetes cluster. The operator pattern in Kubernetes is an extremely powerful extensibility mechanism that allows developers to add additional functionality to the k8s platform, but in a way that it is viewed and managed as part of the platform itself. Operators consist of both a Custom Resource Definition and a controller to manage these resources.

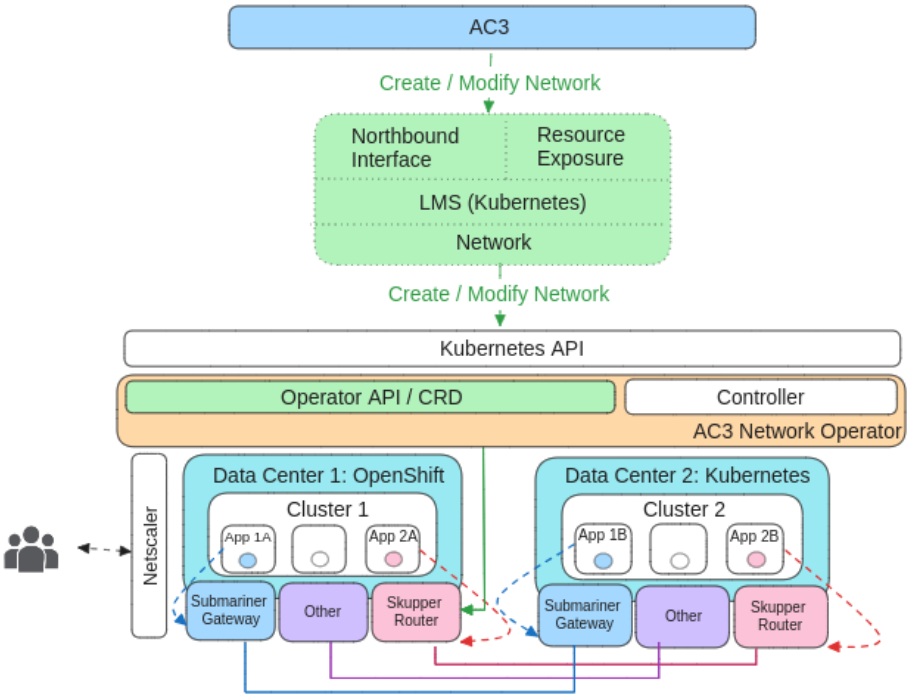

Figure 37: Network management operator

The beauty of this approach is that it allows the central AC³ control plane to manage the network through the same interface with which it can manage any other resource (e.g. provision nodes, deploy applications, etc.). However, this operator provides AC³ an abstraction layer to the underlying Kubernetes network technologies,

meaning AC³ does not need to be aware of the specific technology delivering the network connectivity. Instead, AC³ simply requests connectivity to be created or modified, and the operator will attempt to execute this based on the available network solution.

We also need to consider that there may be multiple different types of interconnected infrastructures that may want to join an AC³ federation. Some infrastructure providers may have closer commercial ties, and therefore have much more tightly coupled technical and security domains, while others may effectively be discrete clusters. In Kubernetes, these translate as multi-clusters with differing forms of control plane architectures. For instance, some data center providers may share a control plane through technologies like Open Cluster Management (OCM), whereby a single control plane administers different clusters. In this model, these multiple clusters can be managed by a single LMS via a central Kubernetes API, meaning that network interconnectivity is more easily established. With standalone clusters, the benefit of Kubernetes is that there is still a standard management API, meaning all network management tasks are executed in the same way. The core difference being that we need to assume security access is granted across all clusters within the AC³ federation. In reality this means that each network operator needs to have access to the APIs of all other clusters in order to be able to link clusters. We aim to be flexible enough to cater for both these scenarios, where the underlying technology used may vary based on the type of control plane used.

7.2.2.2 Custom Resource Definitions (CRD)

As mentioned above, a key part of this pattern is the Customer Resource Descriptor (CRD) that provides the schema definition (and effectively the API) for the new network management resource. Within this project we aim to leverage existing standards specifications, specifically the NetworkGraph object from the NetJson data interchange format (REF). This provides the decorative description of what the overlay connections between the nodes/applications in the network are required by the AC³ control plane. The creation or modification of this definition triggers the controller to then configure the underlying network technologies to deliver this topology. A simplified NetworkGraph example:

```
{
  "type": "NetworkGraph",
  "label": "Test Network",
  "networkType": "van" # mesh, l3, etc

"links": [
    {
      "source": "cluster1-namespace1",
      "target": "cluster2-namespace1",
      "cost": 1.000
    },
    {
      "source": "cluster1-node1",
      "target": "cluster2-node1",
      "cost": 1.000
    },
    {
      "source": "cluster2-node1",
      "target": "cluster2-node1",
      "cost": 1.000
    }
```

```
  ]
}
```

7.2.2.3 AC3 Network Controller

The controller in the AC³ operator monitors its managed resource descriptors, and based on any change, will attempt to create or modify the state of the actual resources in order to align with the requested state. Specifically the controller will attempt to create the requested links between the nodes (applications) using a specific Kubernetes-based network technology. Depending on the type of network connectivity required, different technologies can be employed.

There are common approaches to delivering virtualised networks within the platform itself. Leveraging the underlying CNI, these approaches offer scalable, flexible, and efficient networking solutions, enabling inter-pod communications within a cluster as well as providing access to external services. For instance nodeports, ingress routers and load balancers. While these can support inter-cluster communication, they are typically oriented towards intra-cluster communications, or enabling external access to pods on an individual basis. This makes them not well suited as a comprehensive multi-cluster solution, with issues such as limited protocol support, limited port ranges, limited scalability and performance, as well as security. However there are also many powerful 3rd party networking solutions (for example Submariner and Skupper) that offer additional networking features and capabilities, which utilise the extensibility mechanisms of Kubernetes to integrate seamlessly into the platform. Therefore the technologies currently in scope are Skupper for VAN overlays, and Submariner for layer 3 overlays. For instance, based on a required connection between 2 services, each deployed in a container/pod and hosted in different clusters, the AC³ network operator can configure Skupper to create application specific links between these 2 services.

7.2.2.4 Proactive Network Configuration mechanisms

One of the core aspects of the AC³ network operator is to provision and configure networks between clusters, in order to support the deployment and management of AC³ applications. This will be done based on instruction from AC³'s Decision Enforcement component, via the Adaptation and Federation layer. As mentioned in the previous section, the CRD effectively acts as an interface for AC³ to instruct the network operator to create or modify the network, which will subsequently trigger the controller to create or modify Skupper or Submariner resources. *An overview of Skupper and Submariner is presented in D2.3, however here we will go into more detail on the specific elements of these technologies that the AC³ Network Operator will configure and modify in order to create the required overlays.*

7.2.2.4.1 Skupper Overview

In a Skupper network, the connections between Skupper routers are secured with mutual TLS using a private, dedicated Certificate Authority (CA). Each router is uniquely identified by its own certificate. Skupper ensures security through TLS encryption, mutual TLS authentication and Role Based Access Control (RBAC). These safeguards help maintain the confidentiality and integrity of communication while ensuring only authorised services can interact with each other. Additionally, it's worth noting that Skupper links are namespace-specific, providing isolation and control over communication within individual namespaces.

7.2.2.4.2 Skupper Network Configuration

The process of installing and configuring Skupper in order to link different namespaces creates a number of Kubernetes resources. The core components are:

- **Skupper Router:** Apache Qpid Dispatch Router that handles the actual routing of messages between services across clusters.
- **Site Controller:** Manages the configuration of the scupper site, including artefacts such as configmaps and tokens.
- **Skupper Service Controller:** Handles setup and maintenance of inter-cluster connections, updates routing configuration dynamically when changes occur, exposes local cluster services to other clusters, manages network topology and keeps track of which services are where and how they are connected.

The deployment of the Skupper resources is typically done via CLI, but can also be done directly via YAML files. The sample below demonstrates a declarative descriptor for a Skupper router. We can leverage this to automatically create routers based on the connectivity required by AC^3 and expressed in the network operator CRD.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: skupper-router
spec:
  replicas: 1
  selector:
    matchLabels:
      app: skupper-router
  template:
    metadata:
      labels:
        app: skupper-router
    spec:
      containers:
      - name: skupper-router
        image: quay.io/skupper/edge
        args: ["edge", "start", "--controller", "tcp://skupper-controller:45671"]
        ports:
        - containerPort: 55671
          name: edge
        - containerPort: 45671
          name: controller
```

There are also other advanced configurations and settings that we can exploit. For instance, Skupper can support service load balancing configurations, adjusting traffic distribution across multiple instances when service performance is impacted, thus ensuring constant communication between services across Kubernetes clusters.

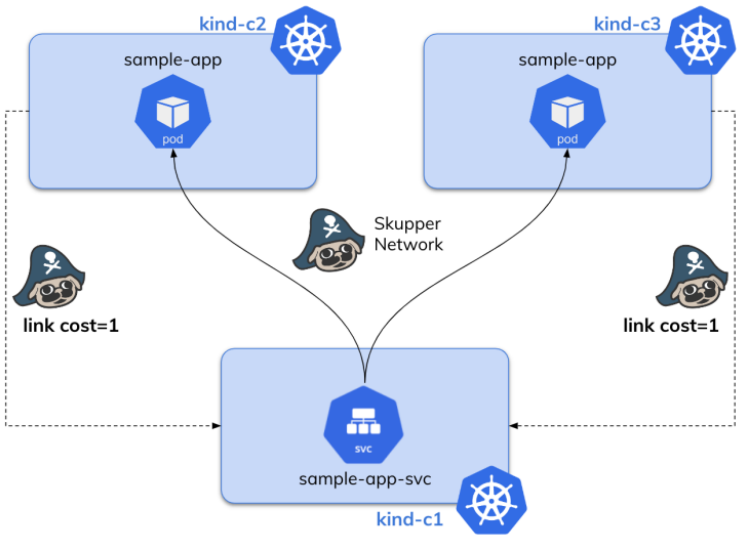

Figure 38: Skupper configuration

When Skupper is configured in a specific namespace, we can then create links from that namespace to namespaces in other clusters. The process of linking 2 namespaces centers around the creation of a token which stores the credentials such as the CA certificates, client certificate and key, and a connection URL which points to the AMQP router on the originating site. The router on the second site is configured with the contents of this token to establish a communication channel, apply routing rules, and synchronize network topology. While some resources like routers can be configured automatically, others such as the sharing of tokens to establish links require manual setup. This is an area we aim to extend Skupper in the context of AC^3 by making this an automated process, using the AC^3 network operator to create the required tokens in other clusters using the K8s API.

7.2.2.4.3 Submariner Network Configuration

Following the same basic approach as above, we can also create Submariner configuration resources in order to create and modify Submariner-based network configuration. This involves installing Submariner components across Kubernetes clusters, defining CRDs for managing connectivity and service discovery, setting parameters, establishing secure connections between clusters, testing communication between workloads and maintaining ongoing monitoring and integration with network plugins. In doing so, having to tailor the configuration to specific requirements ensures effective deployment and management of Submariner-based network connectivity.

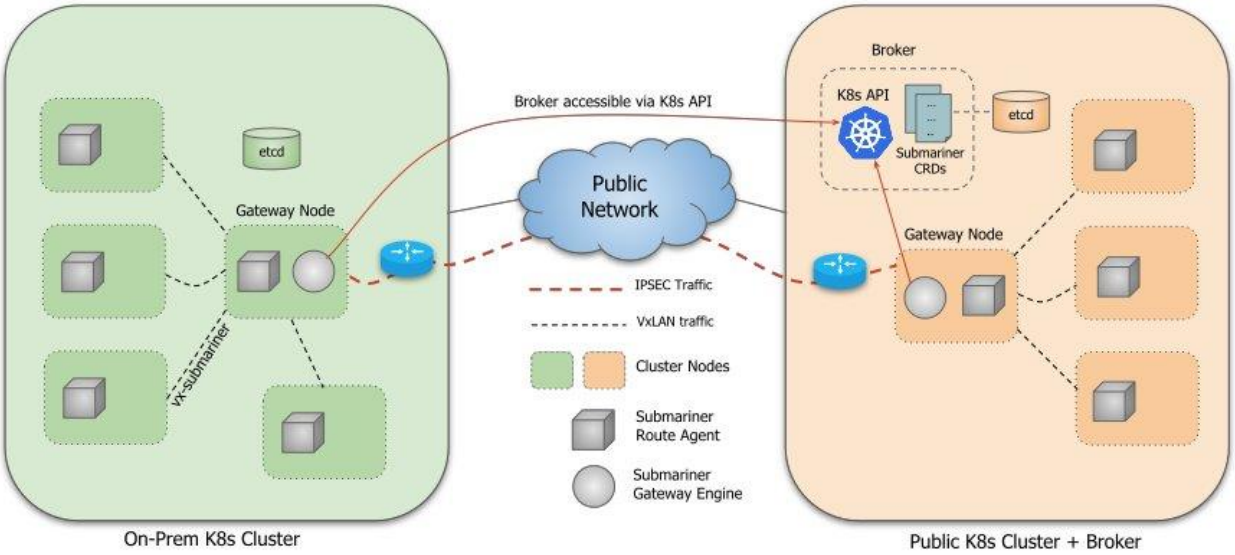

Figure 39: Submariner configuration

The main Submariner architectural components are shown in Figure 39. Of specific interest are the Route Agent, Gateway Engine and Broker. Route Agents are responsible for managing the routing tables and for routing inter-cluster traffic from the node to the cluster's active Gateway Engine, who then sends the traffic to the destination cluster. The Gateway Engine is responsible for creating and maintaining the IPSec tunnels between clusters. The Broker is a central component, installed on a single cluster, that facilitates the exchange of meta-data (specifically CRDs) between clusters. These CRDs, such as endpoint and cluster, contain information about the active Gateway Engine in a cluster, and static information about the originating cluster, such as its service and pod CIDRs.

Unlike Skupper, where each namespace needs to be specifically linked, in Submariner the cross-cluster connection is at the cluster level. So when we join a cluster to the network, a gateway is automatically provisioned, and services within the network are then visible to any connected cluster.

7.2.2.5 Reactive Network Configuration Mechanisms

In the AC^3 network context, reactive network configuration involves dynamically adjusting network settings based on real-time conditions to improve the performance and resource allocation for applications in a Kubernetes cluster. A key call out within this block of work is that we are dealing with virtualized overlay networks, which provide an abstraction on the underlying physical network. As a result, we are limited in the level of fine-grained control we can exert on the network in terms of routing, traffic classification and prioritization, etc. However, within AC^3 we are currently exploring several mechanisms that allow us to react to undesirable conditions within the network and improve the QoS experienced. These approaches are outlined below.

7.2.2.5.1 Multi-overlay Networks

This approach involves dynamically creating multiple overlay networks between applications or clusters in order to distribute network traffic more efficiently, thereby increasing network performance. Within this approach there are 2 potential areas of investigation, load balancing and traffic segmentation.

Load balancing will create multiple overlays with the aim of evenly distributing traffic between them. That is to say that the distribution of traffic is not based on any type of traffic classification. This enhances network performance by dynamically balancing traffic based on factors like network conditions and application

requirements. Traffic segmentation then involves creating and assigning dedicated overlays to specific types of traffic based on predefined criteria. By segmenting traffic, the network operator can ensure that certain types of traffic receive preferential treatment. For instance, traffic with higher latency requirements, such as voice or video, can be segmented into a set of overlays, with a suitable set of resources applied.

In Skupper the primary components responsible for managing the tunnel and routing traffic are the router and the controller. When establishing a connection between 2 namespaces, two routers are created per namespace, these include an Apache Qpid dispatch router which handles the actual routing of messages between services, and a sidecar router responsible for tasks like collecting metrics and monitoring. Whereas the controller itself orchestrates the deployment, configuration and monitoring of the Skupper components. Namespaces can be allocated on a per-application basis but also can be shared by multiple applications, subject to the organization's policies. So, there is a clear performance concern where multiple high-throughput applications can be communicating across a single VAN link, or single extremely high-throughput applications would have only a single router on a given node. Our aim then is to dynamically create multiple Skupper routers per namespace, subject to network conditions, and to then balance requests across these overlays. By creating additional router pods we are also allocating more CPU/memory resources to routing. Figure 40 shows a sample architecture for this multi-router overlay. The broker-based approach is a common mechanism to balance traffic between routers, but also creates a bottleneck at the broker. We will also examine non-broker-based approaches to maximize resilience.

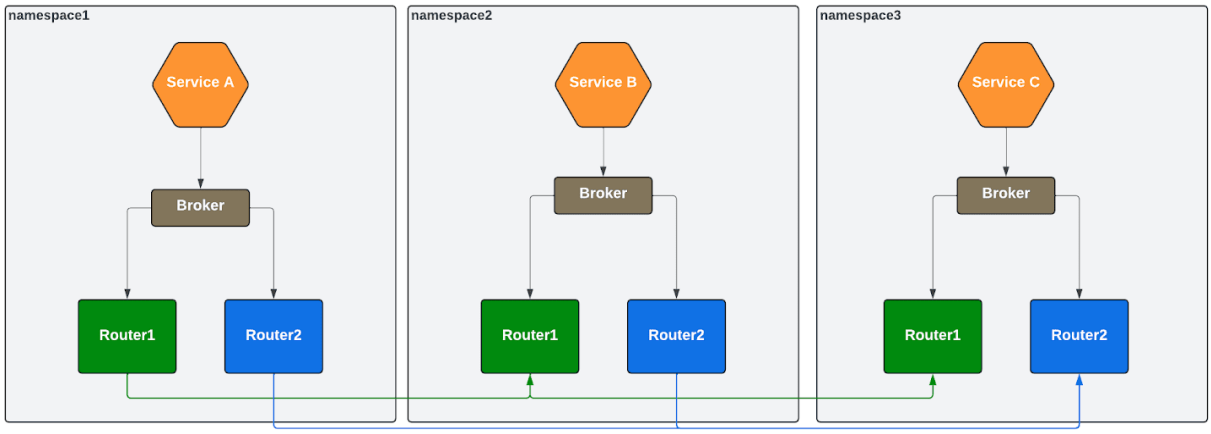

Figure 40: Multiple skupper routers

In Submariner, Gateways are the primary mechanism of establishing and controlling the virtual tunnels between clusters. These tunnels are at the cluster (site) level, meaning all applications in a given cluster will share the same connection. This presents a potential performance bottleneck, whereby high-throughput or high-bandwidth applications could negatively impact the overall performance of all applications on the same cluster. Our proposal is to establish multiple gateways per cluster and allow for balanced routing between them. This also increases the resilience of the Submariner network, as if a single active gateway fails in the current implementation, work must be done to switch over to a new gateway, including making new tunnels and new routes, which takes time.

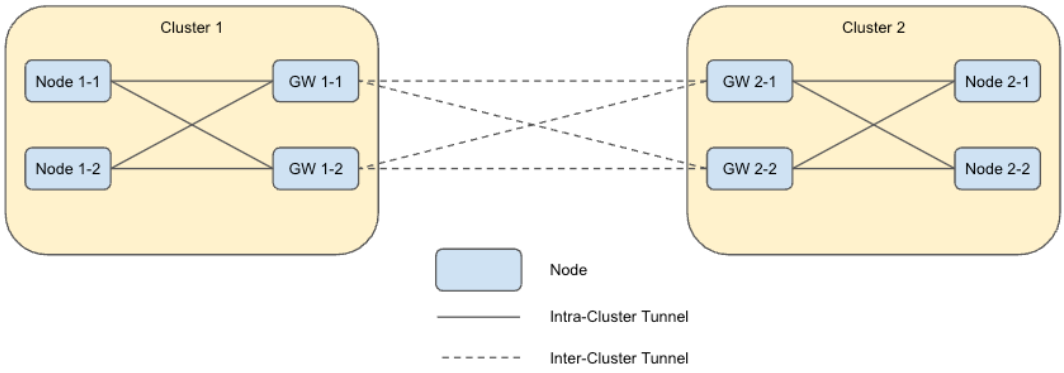

Figure 41: Multiple gateways per cluster

7.2.2.5.2 Adaptive Routing Resource Allocation

One of the major benefits of Kubernetes is that the declarative descriptions of resources can be modified in real-time to adapt to changing conditions within the platform. When we provision network resources that provide specific network functions, we can also adjust these as needed to react to performance concerns. As an example, the Skupper router, responsible for routing traffic from individual namespaces between clusters, creates a Kubernetes pod with a CPU and memory allocation. These allocations can be dynamically updated, and the pod restarted in order to improve the performance of the router.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "skupper-site"
data:
  name: "my-site"
  router-cpu: 2
```

We can also directly modify the deployment description for a given router container to alter the requested CPU and memory as well as their limits.

```
resources:
    requests:
      memory:                                    "64Mi"
      cpu:                                       "250m"
    limits:
      memory:                                    "128Mi"
      cpu: "500m"
```

7.2.2.5.3 Service Load Balancing

One clear mechanism to allow us to impact the experienced QoS is to modify the load balancing configuration of the services, in order to remove load on individual (virtual) network interfaces, network links, etc. This can take several forms:

- Provisioning new pods on different nodes and configuring/modifying load balancing rules.
- Provisioning new Pods on different clusters and configuring/modifying load balancing rules.

Both Skupper and Submariner offer features that can be used to support this load balancing. For instance, Skupper has visibility across clusters as to both service load and available capacity. So, where services are replicated in multiple locations, Skupper can direct requests to specific instances based on available capacity. The AC^3 network operator can then leverage this feature to dynamically modify the deployment location of the services and update the Skupper configuration accordingly, such that load-balancing happens automatically. Furthermore, a cost can be associated with individual links such that we can tailor preferential links.

However, a key consideration is that any alteration of the service placement in the context of load balancing must be done with consideration for the overall service placement strategy as derived by the AC^3 LCM. That is, modifying the load balancing strategy should not violate the service placement decisions made to optimize the resource usage or energy efficiency of the overall system. The link cost feature is something that could be utilized in this regard, assigning link weightings in line with the core placement strategy in order to mitigate significant violations.

7.2.2.5.4 Rate Limiting

Rate limiting is a common approach to throttle the level of incoming requests to a given service in order to reduce certain performance on the service. When we identify a performance issue within a specific network link, we can implement rate limiting to safeguard the available resources. For instance, we could configure the ingress load balancer (NetScaler) to limit the number of requests from a specific location or to a specific application (URL). This is the least desirable approach to managing network performance as, while it preserves the integrity of the service, it has the potential to impact the overall user experience if requests are dropped by throttling. This approach should be done in conjunction with an appropriate service load balancing strategy.

7.2.2.6 Network Monitoring

Another required capability for the Network Programmability component is to expose resources and their metrics to the central AC^3 Resource Discovery and Monitoring components. Also, by proactively monitoring the network, the AC^3 network operator can detect and respond to any anomalies or issues that may arise, allowing for timely mitigations to be implemented and ensuring a reliable network for applications running within the cluster.

In this work we will simply leverage the industry standard Kubernetes monitoring approach, which is typically performed by the Prometheus platform, due to its seamless integration and powerful feature set. In particular its customisable application and system metrics capturing, expressive query language, metrics API and alerting framework. Another important feature is its extensibility. For example the Node Exporter extension allows metrics for the underlying nodes hardware to be gathered.

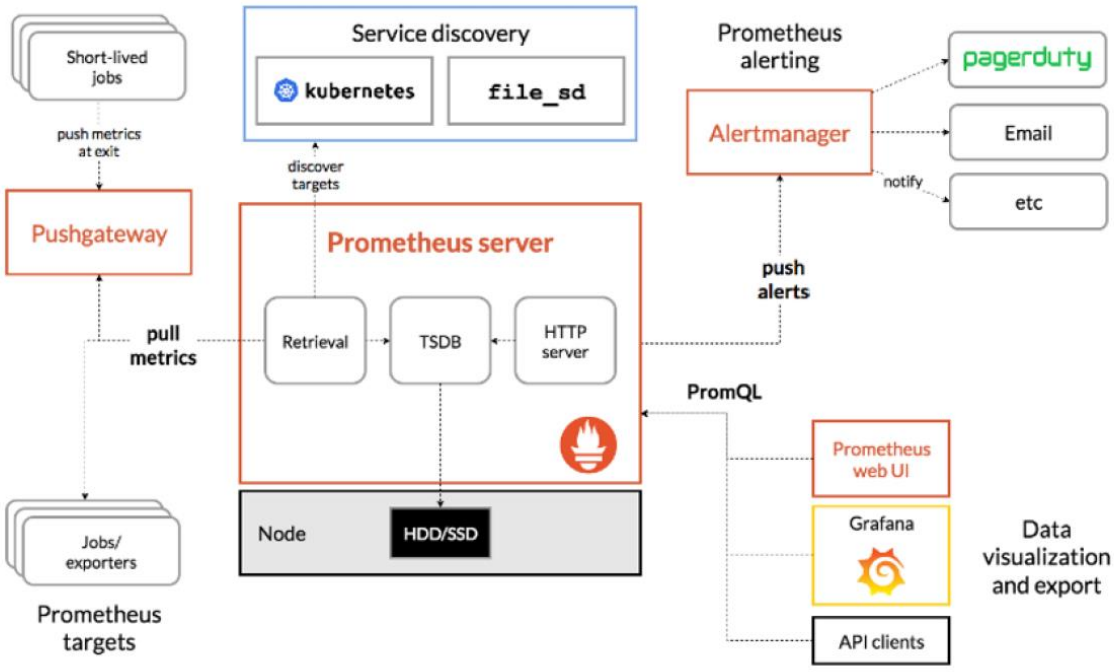

Figure 42: Network monitoring

One of the key tasks going forward is to identify the common set of metrics that all network LMS's should monitor and return to the monitoring component. Within the context of the AC^3 network, this would cover values such as latency, throughout and connection stats.

8 Conclusions

In this deliverable, we presented the initial work pertaining to WP4. We emphasized the progress made in the development of mechanisms for resource discovery and monitoring, Section 4. This includes a unified monitoring data model, emphasizing the need for a comprehensive schema that captures dynamic resource states (CPU, memory, storage), classifies resources by type (edge, cloud), tracks energy sources, and determines geographical locations. We also presented our mechanism for collecting the monitored data and explained the data collection processes for both computing and networking. We provided the operational and application-specific metrics required for the AI-based LCM. This sets the foundation for obtaining the data emanating from the federated infrastructure and for building the resource management capabilities allowing for zero-touch management and configuration, Section 5.

To materialize the promise of AI-based LCM in the context of CECC, several new and innovative AI/ML approaches have been developed, Section 5. The first approach is based on XAI for fine-grained resource autoscaling. This approach combines an ML mechanism, XGBoost, with an explainer, SHAP, for explainable detection of SLA non-compliance and implementation of autoscaling procedures, Section 5.3.2. The second approach uses the Temporal Fusion Transformer for explainable prediction of latency, Section 5.3.3, complementing the mechanism based on XGBoost and presented in Section 5.3.2. The third approach employs a spatio-temporal GNN for resource prediction, particularly workload prediction for volatile nodes using dynamic GNNs, 5.3.4. To consider the prediction results and implement autoscaling procedures, we develop a resource management strategy using RL mechanisms based on a Soft Actor-Critic (SAC) agent, Section 6. To allow the traffic of microservices across the CECCM, a hybrid architecture for multi-layer Network Programmability has been developed in Section 7, where we consider technologies such as Skupper and Submariner.

With regard to Network Programmability, we focus on Software Defined Networking (SDN) and Container-Based Networking (CBN) (Kubernetes), where we provide the possibility for proactive and reactive network configuration. We adapted the SD-WAN architecture by reimplementing certain modules, such as the overlay manager in the SD-WAN controller and the different handlers in the SD-WAN edge. Additionally, we added necessary modules like application-driven routing and monitoring, and we considered an overlay monitoring to provide network metrics to the CECCM resource monitoring module, which can be utilized by an AI-based traffic engineering module. Regarding CBN, we adopted Kubernetes-based network orchestration by designing and building a custom AC^3 network operator. This operator allows to manage and control network configuration within a given Kubernetes cluster. We also proposed a mechanism for multi-overlay networks, involving dynamically creating multiple overlay networks between applications or clusters to distribute network traffic more efficiently, thereby increasing network performance. We provided a capability which by proactively monitoring the network allows the AC^3 network operator to detect and respond to any anomalies or issues that may arise, allowing for timely mitigations to ensure a reliable network for applications running within the cluster, Section 7.2.2.6.

As for future work, whose results will be reported in D4.2, we will be extending the work presented in this deliverable to consider the energy aspect in the AI-based LCM of microservices. Moreover, we will present the operationalization and performance of the monitoring system and discuss its scalability in dealing with various use cases (T4.1). Using the data emanating from the monitoring system, we will report the metrics pertaining to resource and energy usage predictions (T4.2). Specifically, we will present the reduction of SLA non-compliance in an application due to AI/ML resource management capabilities. We will also provide the knowledge model that will be developed to better understand the resource usage and availability (T4.2). We will highlight the performance of the algorithms for the placement, migration, and duplication of microservices on top of the CECC federated infrastructure (T4.3). Additionally, we will report the performance of the Network Programmability approach through its operationalization using SDN and CBN. Key metrics such as decreased downtime caused by

traffic redirection due to Network Programmability and its influence on SLA non-compliance will be provided. Finally, we will show that the implementation of the Network Programmability mechanism will reduce the time needed to update network resources (T4.4).

9 References

[1] https://github.com/netscaler/netscaler-adc-metrics-exporter

[2] https://docs.netscaler.com/en-us/citrix-adc/current-release/observability/prometheus-integration.html

[3] https://docs.tigera.io/calico/latest/about

[4] https://submariner.io/operations/monitoring/

[5] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune and J. Wilkes, "Large-scale cluster management at Google with Borg," In *Proceedings of the 10th European Conference on Computer Systems (EuroSys '15). Association for Computing Machinery*, New York, NY, USA, Article 18, 1–17.

[6] M. Usman, S. Ferlin, A. Brunstrom and J. Taheri, "A Survey on Observability of Distributed Edge & Container-Based Microservices," In *IEEE Access*, vol. 10, pp. 86904-86919, 2022, doi: 10.1109/ACCESS.2022.3193102.

[7] S. Jhingran and N. Rakesh, "Application Deployment and Performance Measurement in Serverless Cloud for Microservices," *In 2023 International Conference on Sustainable Emerging Innovations in Engineering and Technology* (ICSEIET).

[8] T. Khan, W. Tian, G. Zhou, S. Ilager, M. Gong and R. Buyya, "Machine learning (ML)-centric resource management in cloud computing: A review and future directions," In *Journal of Network and Computer Applications*. Volume 204, 2022, 103405, ISSN 1084-8045.

[9] H. Mao, M. Alizadeh, I. Menache and S. Kandula, "Resource Management with Deep Reinforcement Learning," In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16). Association for Computing Machinery,* New York, NY, USA, 50–56.

[10] N. Roy, A. Dubey and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload Forecasting" In *4th International Conference on Cloud Computing*, 2011, pp. 500–507.

[11] M. Abdullah, W. Iqbal, A. Erradi and F. Bukhari, "Learning predictive autoscaling policies for cloud-hosted microservices using trace-driven modeling," In *IEEE International Conference on Cloud Computing Technology and Science* (CloudCom), Sydney, Australia, December 11-13, 2019. IEEE, 2019, pp. 119–126.

[12] T. S. Janjanam, K. S. Siram and P. K. Kollu, "Cloud resources forecasting based on server workload using ml techniques". In *International Conference on Intelligent Data Communication Technologies and Internet of Things* (IDCIoT), 2023, pp. 427-433.

[13] I. K. Kim, W. Wang, Y. Qi and M. Humphrey, "Forecasting cloud application workloads with cloudinsight for predictive resource management," In *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1848–1863, 2022.

[14] T. Mehmood, S. Latif and S. Malik, "Prediction of cloud computing resource utilization,". 2018 In *15th International Conference on Smart Cities: Improving Quality of Life Using ICT IoT* (HONET-ICT), 2018, pp. 38–42.

[15] M. Borkowski, S. Schulte and C. Hochreiner, "Predicting cloud resource utilization," In *IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, 2016, pp. 37–42.

[16] P. Nawrocki, P. Osypanka and B. Posluszny, "Data-driven adaptive prediction of cloud resource usage," *J. Grid Comput.,* vol. 21, no. 1, jan 2023.

[17] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu and H. Tenhunen, "Energy-aware vm consolidation in cloud data centers using utilization prediction model," In *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 524–536, 2019.

[18] G. Christofidi, K. Papaioannou and T. D. Doudali, "Toward pattern-based model selection for cloud resource forecasting," In *Proceedings of the 3rd Workshop on Machine Learning and Systems, ser*. EuroMLSys' 23. New York, NY, USA: Association for Computing Machinery, 2023, p. 115–122.

[19] G. Christofidi, K. Papaioannou and T. D. Doudali, "Is machine learning necessary for cloud resource usage forecasting?" *ACM Symposium on Cloud Computing*, ser. SoCC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 544–554.

[20] J. Bi, S. Li, H. Yuan and M. Zhou, "Integrated deep learning method for workload and resource prediction in cloud systems," In *Neurocomputing*, vol. 424, pp. 35–48, Feb. 2021, publisher Copyright: © 2020 Elsevier B.V.

[21] D. B. Prats, J. L. Berral, C. Wang and A. Youssef, "Proactive container auto-scaling for cloud native machine learning services," In *IEEE 13th International Conference on Cloud Computing* (CLOUD), pp. 475–479, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID: 229358051.

[22] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," *26th Symposium on Operating Systems Principles*, *ser*. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 153–167.

[23] M. Daraghmeh, A. Agarwal, R. Manzano and M. Zaman, "Time series forecasting using facebook prophet for cloud resource managemen," In *IEEE International Conference on Communications Workshops* (ICC Workshops), 2021, pp. 1–6.

[24] M. E. Karim, M. M. S. Maswood, S. Das and A. G. Alharbi, "Bhyprec: A novel bi-lstm based hybrid recurrent neural network model to predict the cpu workload of cloud virtual machine," IEEE Access, vol. 9, pp.131 476–131 495, 2021.

[25] W. Shu, F. Zeng, Z. Ling, J. Liu, T. Lu and G. Chen, "Resource demand prediction of cloud workloads using an attention-based gru model," In *17th International Conference on Mobility, Sensing and Networking* (MSN), 2021, pp. 428–437.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need," In *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[27] Z. Zhou, C. Zhang, L. Ma, J. Gu, H. Qian, Q. Wen, L. Sun, P. Li and Z. Tang, "Ahpa: adaptive horizontal pod autoscaling systems on alibaba cloud container service for Kubernetes". In *Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence, ser.* AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023.

[28] S. Xue, C. Qu, X. Shi, C. Liao, S. Zhu, X. Tan, L. Ma, S. Wang, S. Wang, Y. Hu, L. Lei, Y. Zheng, J. Li and J. Zhang, "A meta reinforcement learning approach for predictive autoscaling in the cloud," In *28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, ser*. KDD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 4290–4299. [Online]. Available: https://doi.org/10.1145/3534678.3539063.

[29] Z. Wang, S. Zhu, J. Li, W. Jiang, K. K. Ramakrishnan, Y. Zheng, M. Yan, X. Zhang and A. X. Liu, "Deepscaling: microservices autoscaling for stable cpu utilization in large scale cloud systems," In *13th Symposium on Cloud Computing*, ser. SoCC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 16–30.

[30] B. Lim, Bryan and S. Zohren, "Time-series forecasting with deep learning: a survey. Philosophical Transactions". In *Series A, mathematical, physical, and engineering sciences*. 379. 20200209. 10.1098/rsta.2020.0209.

[31] M. Mekki, B. Brik, A. Ksentini and C. Verikoukis, "XAI-Enabled Fine Granular Vertical Resources Autoscaler," In *9th International Conference on Network Softwarization* (NetSoft), Madrid, Spain, 2023, pp. 161-169, doi: 10.1109/NetSoft57336.2023.10175438.

[32] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser.* KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794.

[33] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions". In *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[34] ENI Vision: Improved Network Experience using Experiential Networked Intelligence, ETSI ENI White paper.
[35] Network Functions Virtualization (NFV) Release 3; Architecture. *Report on the Enhancements of the NFV architecture towards Cloud-native and PaaS*. ETSI GR NFV-IFA 029 V3.3.1, Nov. 2019.
[36] M. T. Ribeiro, S. Singh and C. Guestrin, "Why should i trust you? Explaining the predictions of any classifier," In *22nd ACM SIGKDD international conference on knowledge discovery and data mining,* 2016, pp. 1135–1144.
[37] B. Lim, S. Arık, N. Loeff and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," In *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748– 1764, 2021.
[38] A. Vaswani, N.Shazeer, N.Parmar, J.Uszkoreit, L.Jones, A.N.Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need". In *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.
[39] H. He, L. Su and K. Ye, "Graphgru: A graph neural network model for resource prediction in microservice cluster," In *28th International Conference on Parallel and Distributed Systems* (ICPADS), Nanjing, China, 2023, pp. 499-506, doi: 10.1109/ICPADS56603.2022.00071. pp. 499–506, 2023.
[40] A. Sankar, Y. Wu, L. Gou, W. Zhang and H. Yang, "Dynamic graph representation learning via self-attention networks," In *ArXiv* abs/1812.09430 (2018).
[41] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," In *ArXiv* abs/1902.10191 (2019): n. pag.
[42] J. Dogani, R. Namvar and F. Khunjush, "Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey," In *Computer Communications*, pp. 120-150, Sep. 2023.
[43] J. Dogani, F. Khunjush, M.R. Mahmoudi and M. Seydali, "Multivariate workload and resource prediction in cloud computing using CNN and GRU by attention mechanism," In *The Journal of Supercomputing*, pp. 3437–3470, 2023.
[44] C. Zhu, B. Han and Y. Zhao, "A bi-metric autoscaling approach for n-tier web applications on Kubernetes," In *Frontiers of Computer Science*, pp. 1–12, 2022.
[45] M. Goudarzi, M.S. Palaniswami and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," In *IEEE Transactions on Mobile Computing*, pp. 2491-2505, 2021.
[46] Y. Chen, "Reinforcement Learning Meets Wireless Networks: A Layering Perspective," In *IEEE Internet of Things Journal*, pp. 85–111, 2021.
[47] A. Kaloxyos, A. Gravras, D. Camps Mur, M. Ghoraishi, "AI and ML – Enablers for Beyond 5G Networks," In https://zenodo.org/records/4299895.
[48] Z. Zhang, T. Wang, A. Li and W. Zhang, "Adaptive auto-scaling of delay-sensitive serverless services with reinforcement learning," In *46th IEEE Annual Computers, Software, and Applications Conference* (COMPSAC), pp. 866–871, 2022.
[49] P. Benedetti, M. Femminella, G. Reali and K. Steenhaut, "Reinforcement learning applicability for resource-based auto-scaling in serverless edge applications," In *IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events*, pp. 674–679, 2022.
[50] G. Somma, C. Ayimba, P. Casari, S. P. Romano and V. Mancuso, "When less is more: Core-restricted container provisioning for serverless computing," In *IEEE Conference on Computer Communications Workshops*, pp. 1153–1159, 2020.
[51] S. Agarwal, M. A. Rodriguez and R. Buyya, "A reinforcement learning approach to reduce serverless function cold start frequency," In *21st ACM International Symposium on Cluster, Cloud and Internet Computing*, pp. 797–803, 2021.

[52] A. Zafeiropoulos, E. Fotopoulou, N. Filinis and S. Papavassiliou, "Reinforcement learning-assisted autoscaling mechanisms for serverless computing platforms," In *Simulation Modelling Practice and Theory*, vol. 116, 2022.
[53] P. Dai, K. Hu, X. Wu, H. Xing and Z. Yu, "Asynchronous Deep Reinforcement Learning for Data-Driven Task Offloading in MEC Empowered Vehicular Networks," In *IEEE Conference on Computer Communications*, pp. 1-10, 2021.
[54] X. Chen, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning," In *IEEE Internet of Things Journal*, vol.6, no.3, pp. 4005–4018, Oct. 2018.
[55] Z. Ning, "When Deep Reinforcement Learning Meets 5G-Enabled Vehicular Networks: A Distributed Offloading Framework for Traffic Big Data," In *IEEE Transactions on Industrial Informatics*, vol.16, no. 2, pp. 1352–1361, 2020.
[56] KH. Liu and W. Liao, "Intelligent Offloading for Multi-Access Edge Computing: A New Actor-Critic Approach," In *IEEE International Conference on Communications* (ICC), pp. 1–6, 2020.
[57] X. Chen, C. Wu, Z. Liu, N. Zhang and Y. Ji, "Computation Offloading in Beyond 5G Networks: A Distributed Learning Framework and Applications," In *IEEE Wireless Communications,* vol. 28, no. 2, pp. 56-62, Apr. 2021.
[58] J. Ren, "Federated Learning-Based Computation Offloading Optimization in Edge Computing-Supported Internet of Things," In *IEEE Access* 7, pp. 69194–69201, 2019.
[59] N. Shan, X. Cui and Z. Gao, "DRL + FL": An intelligent resource allocation model based on deep reinforcement learning for Mobile Edge Computing," In *Computer Communications* 160, pp. 14–24, 2020.
[60] 3GPP, "3rd Generation Partnership Project," *Technical Specification Group Services and System Aspects, Service requirements for video, imaging and audio for professional applications* (VIAPA), 2021, 3GPP TS 22.263 V17.4.0 (2021-06).
[61] 3GPP, "3rd Generation Partnership Project," *Technical Specification Group Services and System Aspects, Service requirements for cyber-physical control applications in vertical domains*, 2023, 3GPP TS 22.104 V19.1.0 (2023-09).
[62] D.J. Kerbyson, A. Hoisie, S. Pakin, F. Petrini and H.J. Wasserman, "A Performance Evaluation of an Alpha EV7 Processing Node," In *The International Journal of High Performance Computing Applications*, pp. 199-209, 2004.
[63] T. Haarnoj, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," In *ICML*, 2018.
[64] L. A. Garrido, K. Ramantas, A. Dalgkitsis, A. Ksentini and C. Verikoukis, "Admission Control with Resource Efficiency Using Reinforcement Learning in Beyond-5G Networks" *In 34th IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications* (PIMRC), pp. 1-6, 2023.
[65] Python, "https://www.python.org/,". Online: accessed 31 March 2024.
[66] Tensorflow, "https://www.tensorflow.org/,". Online: accessed 31 March 2024.
[67] Google cluster-usage traces v3, "https://github.com/google/clusterdata/blob/master/ClusterData2019.md,". Online: accessed 31 March 2024.
[68] W.A. Hanafy, "CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency," In *ACM on Measurement and Analysis of Computing Systems*, pp. 1-28, 2023.
[69] N. Gholipour, E. Arianyan and R. Buyya, "A novel energy-aware resource management technique using joint VM and container consolidation approach for green computing in cloud data centers," In *Simulation Modelling Practice and Theory*, 2020.
[70] M. Awad, N. Kara and C. Edstrom, "SLO-aware dynamic self-adaptation of resources," In *Future Generation Computer Systems*, pp. 266-280, 2022.

[71] G. Nieto, I. de la Iglesia, U. López-Novoa and C. Perfecto, "Deep Reinforcement Learning-based Task Offloading in MEC for energy and resource-constrained devices," In *IEEE International Mediterranean Conference on Communications and Networking* (MeditCom), pp. 127-132, 2023.
[72] Y. Xiao, Y. Song and J. Liu, "Towards Energy Efficient Resource Allocation: When Green Mobile Edge Computing Meets Multi-Agent Deep Reinforcement Learning,". In *IEEE International Conference on Communications*, pp. 4056-4061, 2022.
[73] M. Qin, L. Chen, N. Zhao, Y. Chen, F. R. Yu and G. Wei, "Power-Constrained Edge Computing with Maximum Processing Capacity for IoT Networks," In *IEEE Internet of Things Journal,* vol. 6, no. 3, pp. 4330-4343, Jun. 2019.
[74] M. Song, Y. Lee and K. Kim, "Reward-Oriented Task Offloading Under Limited Edge Server Power for Multiaccess Edge Computing," In IEEE *Internet of Things Journal,* vol. 8, no. 17, pp. 13425-13438, Sep. 2021.
[75] K. Phemius and M. Bouet, "Implementing OpenFlow-based resilient network services," In *IEEE 1st International Conference on Cloud Networking* (CLOUDNET).
[76] A. Fressancourt and M. Gagnaire, "A SDN-based network architecture for cloud resiliency," In *12th Annual IEEE Consumer Communications and Networking Conference* (CCNC).
[77] E. Mora-Huiracocha, "Implementation of a SD-WAN for the interconnection of two software defined data centers," In IEEE *Colombian Conference on Communications and Computing* (COLCOM).
[78] J. Sushant., "B4: Experience with a globally-deployed software defined WAN," In *ACM SIGCOMM Computer Communication Review* 43.4 (2013): 3-14.
[79] H. Chi-Yao, "B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN," In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*.
[80] F. Bannour, S. Souihi and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," In *IEEE Communications Surveys & Tutorials* 20.1 (2017): 333-354.
[81] H. Chi-Yao, "Achieving high utilization with software-driven WAN," In *Proceedings of the ACM SIGCOMM 2013 Conference*.
[82] T. Sebastian, "SD-WAN: an open-source implementation for enterprise networking services," In *22nd International Conference on Transparent Optical Networks* (ICTON). IEEE, 2020.
[83] J. Medved, "Opendaylight: Towards a model-driven sdn controller architecture," In *Proceeding of IEEE international symposium on a world of wireless, mobile and multimedia networks 2014*.
[84] Open vSwitch, "https://www.openvswitch.org," (2024, January 21)
[85] VyOS, "Universal Open Source Router,". https://vyos.io. (2024, December 2).
[86] Open Source SD-WAN and SASE, "https://flexiwan.com" (2023, November 14).
[87] S. Carmine, "EveryWAN-an open source SD-WAN solution," In *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering* (ICECCME).
[88] FRRouting Project, "https://frrouting.org" (2023, December 19).
[89] VPP Technology, "https://fd.io/technology," (2023, December 19).
[90] Pyroute2, "https://pypi.org/project/pyroute," (2023, December 12).
[91] M. Mahalingam, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," No. rfc7348. 2014.
[92] F. Dino, "Generic routing encapsulation (GRE)," No. rfc2784. 2000.

10 Useful links

https://www.citrix.com/blogs/2014/05/02/netscaler-counters-grab-bag/
https://yuminlee2.medium.com/kubernetes-container-network-interface-cni-ee5b21514664
https://docs.tigera.io/calico/latest/about
https://github.com/flannel-io/flannel
https://www.solo.io/topics/cilium/
https://skupper.io/docs/overview/index.html
https://submariner.io/getting-started/architecture/
https://istio.io/latest/docs/concepts/traffic-management/
https://developer.cisco.com/docs/cloud-native-sdwan/
https://www.vmware.com/topics/glossary/content/kubernetes.html
https://submariner.io/operations/monitoring/

11 Annexes

11.1 Annexe A – SD-WAN Northbound API Swagger Definition

```
openapi: "3.0.0"
info:
 title: "SD-WAN Service"
 description: "Northbound REST API of the SD-WAN controller"
 version: "0.1.1"
servers:
 - url: "/"
paths:
 /overlays:
   get:
     summary: "Retrieve Created Overlays"
     description: "Get details of created overlays of a given tenant ID. Can be filtered with
overlays IDs."
     parameters:
       - name: "overlaysIDs"
         in: "query"
         schema:
           type: "array"
           items:
             type: "string"
         description: "List of overlays IDs. For filtering."
       - name: "tenantid"
         in: "query"
         schema:
           type: "string"
           default: "1"
         required: true
         description: "The tenant ID"
     responses:
       200:
         description: "A list of overlays"
         content:
           application/json:
             schema:
               type: array
               items:
                 $ref: '#/components/schemas/OverlayInfo'
       400:
         description: "Invalid request parameters."
       401:
         description: "Unauthorized access."
       500:
         description: "Internal server error."
       503:
         description: "Service unavailable. The server is temporarily unable to handle the request."
   post:
     summary: "Create Overlay"
     description: "Create a new SD-WAN overlay."
     requestBody:
       required: true
       content:
         application/json:
           schema:
```

```
            $ref: "#/components/schemas/OverlayConfig"
      responses:
        200:
          description: "Overlay created successfully."
        400:
          description: "Invalid request payload."
        401:
          description: "Unauthorized access."
        500:
          description: "Internal server error."
        503:
          description: "Service unavailable. The server is temporarily unable to handle the request."
    delete:
      summary: "Remove Overlay"
      description: "Delete an overlay by specifying either overlay name or overlay ID along with
tenant ID."
      requestBody:
        required: true
        content:
          application/json:
            schema:
              oneOf:
                - $ref: "#/components/schemas/OverlayReqID"
                - $ref: "#/components/schemas/OverlayReqName"
      responses:
        200:
          description: "Overlay removed successfully."
        400:
          description: "Invalid request payload."
        401:
          description: "Unauthorized access."
        500:
          description: "Internal server error."
        503:
          description: "Service unavailable. The server is temporarily unable to handle the request."
  /overlays/default:
    post:
      summary: "Create Default Overlay"
      description: "Create a default SD-WAN overlay with predefined default parameters."
      parameters:
        - name: "tenantid"
          in: "query"
          required: true
          schema:
            type: "string"
            default: "1"
      responses:
        200:
          description: "Default overlay created successfully."
        208:
          description: "Default overlay already exists."
        400:
          description: "Invalid request payload."
        401:
          description: "Unauthorized access."
        500:
          description: "Internal server error."
```

```
      503:
        description: "Service unavailable. The server is temporarily unable to handle the request."
/enableServicePublicAccess:
  post:
    summary: "Enable service public access"
    description: "Enable public access to a specific service."
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ServicePublicAccessParams'
    responses:
      200:
        description: Success
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
                public_ip:
                  type: string
                public_port:
                  type: integer
      400:
        description: Bad Request
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
      500:
        description: Internal Server Error
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
/disableServicePublicAccess:
  post:
    summary: "Disable service public access"
    description: "Disable public access to a specific service."
    requestBody:
      required: true
      content:
```

```
        application/json:
          schema:
            $ref: '#/components/schemas/ServicePublicAccessParams'
    responses:
      200:
        description: Success
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
      400:
        description: Bad Request
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
      500:
        description: Internal Server Error
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
/enableServiceInterClusterAccess:
  post:
    summary: "Enable service inter-cluster access"
    description: "Enable inter-cluster access for a specific overlay."
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ServiceInterClusterAccessParams'
    responses:
      200:
        description: Success
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
```

```
                reason:
                  type: string
      400:
        description: Bad Request
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
      500:
        description: Internal Server Error
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
/disableServiceInterClusterAccess:
  post:
    summary: "Disable service inter-cluster access"
    description: "Disable inter-cluster access for a specific overlay."
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ServiceInterClusterAccessParams'
    responses:
      200:
        description: Success
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
      404:
        description: Not Found
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
```

```
      500:
        description: Internal Server Error
        content:
          application/json:
            schema:
              type: object
              properties:
                code:
                  type: integer
                reason:
                  type: string
 /edges/configurations:
   get:
     summary: "Retrieve Edges Interfaces Configurations."
     description: "Fetch the configurations of registered SD-WAN edge devices interfaces. Can be
filtered based on device IDs."
     parameters:
       - name: "devicesids"
         in: "query"
         schema:
           type: "array"
           items:
             type: "string"
         description: "List of devices IDs."
       - name: "tenantid"
         in: "query"
         schema:
           type: "string"
           default: "1"
         required: true
         description: "The tenant ID."
     responses:
       200:
         description: "A list of All edges configurations."
         content:
           application/json:
             schema:
               type: array
               items:
                 $ref: '#/components/schemas/DeviceConfigInfo'
       400:
         description: "Invalid request parameters."
       401:
         description: "Unauthorized access."
       500:
         description: "Internal server error."
       503:
         description: "Service unavailable. The server is temporarily unable to handle the request."
 /edges/interfacesconfiguration:
   post:
     summary: "Configure Edges Interfaces"
     description: "Set up interfaces for SD-WAN edge devices. Configuring the Edges Interfaces is
required to init the SD-WAN controller. This enables the edges. Its for specifying underlay wan
ids, and subnets that are behind each edge (local subnetworks)."
     requestBody:
       required: true
       content:
```

```
        application/json:
          schema:
            $ref: "#/components/schemas/DevicesConfigs"
    responses:
      200:
        description: "Edge interfaces configured successfully."
      400:
        description: "Invalid request payload."
      401:
        description: "Unauthorized access."
      500:
        description: "Internal server error."
      503:
        description: "Service unavailable. The server is temporarily unable to handle the request."
/appIdentifiers:
  get:
    summary: "Retrieve Application Identifiers"
    description: "Get details of application identifiers."
    parameters:
      - name: "tenantid"
        in: "query"
        required: true
        schema:
          type: "string"
          default: "1"
        description: "Tenant ID"
      - name: "appsIdentifIDs"
        in: "query"
        schema:
          type: "array"
          items:
            type: "string"
        description: "List of applications identifiers IDs"
    responses:
      200:
        description: "A list of application identifiers."
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/AppIdentifierInfo'
      400:
        description: "Invalid request parameters."
      401:
        description: "Unauthorized access."
      500:
        description: "Internal server error."
      503:
        description: "Service unavailable. The server is temporarily unable to handle the request."
  post:
    summary: "Create Application Identifier"
    description: "Create a new application identifier."
    requestBody:
      required: true
      content:
        application/json:
```

```
          schema:
            $ref: "#/components/schemas/ApplicationIdentifier"
      responses:
        200:
          description: "Application identifier created successfully."
        400:
          description: "Invalid request payload."
        401:
          description: "Unauthorized access."
        500:
          description: "Internal server error."
        503:
          description: "Service unavailable. The server is temporarily unable to handle the request."
    delete:
      summary: "Remove Application Identifier"
      description: "Delete an application identifier by specifying either application identifier
name or ID along with tenant ID."
      requestBody:
        required: true
        content:
          application/json:
            schema:
              oneOf:
                - $ref: "#/components/schemas/AppIdentifReqID"
                - $ref: "#/components/schemas/AppIdentifReqName"
      responses:
        200:
          description: "Application identifier removed successfully."
        400:
          description: "Invalid request payload."
        401:
          description: "Unauthorized access."
        500:
          description: "Internal server error."
        503:
          description: "Service unavailable. The server is temporarily unable to handle the request."
  /monitoring/overlay/latency/start:
    post:
      summary: "Start Overlay Tunnels Monitoring"
      description: "Initiate the monitoring of all overlay tunnels' latency."
      parameters:
        - name: "tenantid"
          in: "query"
          required: true
          schema:
            type: "string"
            default: "1"
          description: "Tenant ID"
      responses:
        200:
          description: "Overlay tunnels monitoring started successfully."
        400:
          description: "Invalid request parameters."
        401:
          description: "Unauthorized access."
        500:
          description: "Internal server error."
```

```
      503:
        description: "Service unavailable. The server is temporarily unable to handle the request."
/monitoring/overlay/latency/stop:
  post:
    summary: "Stop Overlay Tunnels Monitoring"
    description: "Stop the monitoring of all overlay tunnels' latency."
    parameters:
      - name: "tenantid"
        in: "header"
        required: true
        schema:
          type: "string"
          default: "1"
        description: "Tenant ID"
    responses:
      200:
        description: "Overlay tunnels monitoring stopped successfully."
      400:
        description: "Invalid request parameters."
      401:
        description: "Unauthorized access."
      500:
        description: "Internal server error."
      503:
        description: "Service unavailable. The server is temporarily unable to handle the request."
/monitoring/overlay/latency:
  get:
    summary: "Retrieve Overlays Tunnels latency"
    description: "Get latency for all overlay tunnels."
    parameters:
      - name: "tenantid"
        in: "query"
        required: true
        schema:
          type: "string"
          default: "1"
        description: "Tenant ID"
    responses:
      200:
        description: "A list of overlays tunnels latency data."
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/OverlayLatency'
      400:
        description: "Invalid request parameters."
      401:
        description: "Unauthorized access."
      500:
        description: "Internal server error."
      503:
        description: "Service unavailable. The server is temporarily unable to handle the request."
```

```yaml
components:
 schemas:
   OverlayLatency:
     type: object
     properties:
       overlay_name:
         type: string
         description: "Name of the overlay."
       overlayid:
         type: string
         description: "Unique identifier for the overlay."
       tunnels:
         type: array
         description: "List of tunnels associated with the overlay."
         items:
           $ref: '#/components/schemas/TunnelLatency'
   TunnelLatency:
     type: object
     properties:
       tunnel_key:
         type: string
         description: "Unique key for the tunnel."
       tunnel_interface_name:
         type: string
         description: "Name of the Overlay tunnel interface."
       edge:
         type: string
         description: "Edge That initiated the ping."
       latency_history:
         type: array
         description: "History of latency measurements for the tunnel."
         items:
           type: number
           format: float
   AppIdentifierInfo:
     type: object
     properties:
       device_name:
         type: string
         description: "Name of the edge device in which the rules will be matched."
       tenantid:
         type: string
         description: "Tenant ID."
       id:
         type: string
         description: "Unique identifier for the application Identifier."
       application_name:
         type: string
         description: "Name of the application."
       description:
         type: string
       category:
         type: string
```

```
        description: "Category of the application (e.g., data-transfer)."
      service_class:
        type: string
        description: "Service class of the application (e.g., high-throughput)."
      importance:
        type: string
        description: "Importance level of the application."
        enum: [LOW, MEDIUM, HIGH]
      overlay_paths:
        type: object
        description: "Overlay paths configuration for the application."
        properties:
          paths_mode:
            type: string
            description: "Mode of the paths (e.g., static)."
            enum: [static, dynamic]
          paths:
            type: array
            description: "List of overlay paths."
            items:
              type: string
          policy:
            type: string
            description: "Policy for dynamic overlay paths selection."
          delay_threshold:
            type: number
            description: "Delay threshold."
      rules:
        $ref: "#/components/schemas/APPIdentifRule"
      matches:
        $ref: "#/components/schemas/APPIdentifMatch"
  OverlayInfo:
    type: object
    properties:
      id:
        type: string
        description: "Unique identifier of the overlay."
      name:
        type: string
        description: "Overlay name."
      type:
        type: string
        enum: [IPv4Overlay, IPv6Overlay]
        description: "Type of the overlay."
      underlay_wan_id:
        type: string
        description: "ID of the underlay WAN associated with the overlay."
      mode:
        type: string
        enum: [VXLAN]
        description: "Overlay Tunnel Mode (e.g., VXLAN)."
      tenantid:
        type: string
        description: "Tenant ID."
      interfaces:
        type: array
        description: "List of interfaces associated with the overlay."
```

```
      items:
        $ref: '#/components/schemas/OverlayInterface'
OverlayInterface:
  type: object
  properties:
    device_name:
      type: string
      description: "Device Name."
    deviceid:
      type: string
    interface_name:
      type: string
      description: "Name of the LAN interface associated with Overlay Slice."
DeviceConfigInfo:
  type: object
  properties:
    device_id:
      type: string
      description: "Unique identifier for the edge."
    name:
      type: string
      description: "Name of the edge device."
    description:
      type: string
    mgmtip:
      type: string
      description: "The IP address used for SBI gRPC Server."
    connected:
      type: boolean
      description: "Indicates if the device is registered with the SD-WAN controller."
    configured:
      type: boolean
      description: "Indicates if the device is configured."
    enabled:
      type: boolean
      description: "Indicates if the device is enabled."
    default_underlay_id:
      type: string
      description: "default WAN Id to be used in creating default overlay."
    interfaces:
      type: array
      items:
        $ref: '#/components/schemas/DeviceInterfConfigInfo'
      description: "List of network interfaces for the device."
    loopbackip:
      type: string
      description: "Loopback IP address of the device."
    loopbacknet:
      type: string
      description: "Loopback network of the device."
    managementip:
      type: string
      description: "The IP address used for SBI gRPC Server."
DeviceInterfConfigInfo:
  type: object
  properties:
    name:
```

```
        type: string
        description: "Name of the network interface."
      mac_addr:
        type: string
        description: "MAC address of the network interface."
      ipv4_addrs:
        type: array
        items:
          type: string
        description: "List of IPv4 addresses assigned to the network interface."
      ipv6_addrs:
        type: array
        items:
          type: string
        description: "List of IPv6 addresses assigned to the network interface."
      ext_ipv4_addrs:
        type: array
        description: "List of external IPv4 addresses assigned to the network interface."
        items:
          type: string
      ext_ipv6_addrs:
        type: array
        description: "List of external IPv6 addresses assigned to the network interface."
        items:
          type: string
      ipv4_subnets:
        type: array
        description: "List of IPv4 subnets behind this interface. (If type is lan)."
        items:
          $ref: '#/components/schemas/Ipv4Subnet'
      ipv6_subnets:
        type: array
        description: "List of IPv6 subnets behind this interface. (If type is lan)."
        items:
          $ref: '#/components/schemas/Ipv6Subnet'
      type:
        type: string
        description: "Type of the network interface."
        enum: [lan, wan, unknown]
      underlay_wan_id:
        type: string
        description: "ID of the underlay WAN associated to this network interface."
  DevicesConfigs:
    type: object
    properties:
      tenantid:
        type: string
        description: "Tenant ID."
        example: "1"
      default_underlay_id:
        type: string
        description: "default WAN Id to be used in creating default overlay."
        example: "WAN-1"
      devices_configs:
        type: array
        description: "The list of devices configurations."
        items:
```

```
        $ref: "#/components/schemas/DeviceInterfacesConfig"
      example:
        - device_name: "ewED1"
          interfaces:
            - name: "ew_if1_wan2"
              type: "wan"
              underlay_wan_id: "WAN-1"
            - name: "ew_if2_sboai"
              type: "wan"
              underlay_wan_id: "WAN-2"
            - name: "el_if1_priv_net"
              type: "lan"
              ipv4_subnets:
                - gateway: ""
                  subnet: "192.168.50.0/24"
                - gateway: "192.168.50.9"
                  subnet: "10.168.55.0/24"
        - device_name: "ewED2"
          interfaces:
            - name: "ew_if1_wan2"
              type: "wan"
              underlay_wan_id: "WAN-1"
            - name: "ew_if2_sboai"
              type: "wan"
              underlay_wan_id: "WAN-2"
            - name: "el_if1_priv_net"
              type: "lan"
              ipv4_subnets:
                - gateway: ""
                  subnet: "192.168.70.0/24"
                - gateway: "192.168.70.9"
                  subnet: "10.168.77.0/24"
DeviceInterfacesConfig:
  type: object
  properties:
    device_name:
      type: string
      description: "Device name to configure its interfaces."
    interfaces:
      type: array
      items:
        $ref: "#/components/schemas/InterfaceConfig"
InterfaceConfig:
  type: object
  properties:
    name:
      type: string
      description: "Name of the interface."
    type:
      type: string
      description: "Type of interface (e.g., wan, lan)."
      enum: [lan, wan]
    underlay_wan_id:
      type: string
      description: "ID of the underlay WAN."
    ipv4_addrs:
      type: array
```

```
        items:
          type: string
        description: "List of IPv4 addresses."
      ipv6_addrs:
        type: array
        items:
          type: string
        description: "List of IPv6 addresses."
      ipv4_subnets:
        type: array
        items:
          $ref: "#/components/schemas/Ipv4Subnet"
        description: "List of IPv4 subnets."
      ipv6_subnets:
        type: array
        items:
          $ref: "#/components/schemas/Ipv6Subnet"
        description: "List of IPv6 subnets."
      ext_ipv4_addrs:
        type: array
        items:
          type: string
        description: "List of external IPv4 addresses."
      ext_ipv6_addrs:
        type: array
        items:
          type: string
        description: "List of external IPv6 addresses."
  Ipv4Subnet:
    type: object
    properties:
      subnet:
        type: string
        description: "IPv4 subnet."
      gateway:
        type: string
        description: "Gateway address for the IPv4 subnet."
    description: "IPv4 subnet, and the gateway to this subnet."
  Ipv6Subnet:
    type: object
    properties:
      subnet:
        type: string
        description: "IPv6 subnet."
      gateway:
        type: string
        description: "Gateway address for the IPv6 subnet."
    description: "IPv6 subnet, and the gateway to this subnet."
  OverlayConfig:
    type: object
    properties:
      tenantid:
        type: string
        description: "The ID of the tenant."
        example: "1"
      overlay_name:
        type: string
```

```
        description: "The name of the overlay."
        example: "Overlay-100"
      overlay_type:
        type: string
        description: "The type of the overlay."
        enum: [IPv4Overlay, IPv6Overlay]
        example: "IPv4Overlay"
      overlay_tunnel_mode:
        type: string
        description: "The tunneling mode of the overlay."
        enum: [VXLAN]
        example: "VXLAN"
      overlay_slices:
        type: array
        description: "The slices of the overlay. i.e. the lan interfaces behind the edge that will
be assigned to the overlay."
        items:
          $ref: "#/components/schemas/OverlaySlice"
        example:
          - device_name: "ewED1"
            device_LAN_interface: "el_if1_priv_net"
            undelay_WAN_id: "WAN-1"
          - device_name: "ewED2"
            device_LAN_interface: "el_if1_priv_net"
            undelay_WAN_id: "WAN-1"
    required:     ["tenantid",     "overlay_name",     "overlay_type",     "overlay_tunnel_mode",
"overlay_slices"]
  OverlaySlice:
    type: object
    properties:
      device_name:
        type: string
        description: "Edge device name associated with the slice."
      device_LAN_interface:
        type: string
        description: "The LAN interface of the edge device to be assigned to the slice."
      undelay_WAN_id:
        type: string
        description: "The ID of the underlay WAN interface. The overlay will be created over this
underlay."
    required: ["device_name", "device_LAN_interface", "undelay_WAN_id"]
  ApplicationIdentifier:
    type: object
    properties:
      device_name:
        type: string
        description: "Name of the device."
        example: "ewED1"
      tenantid:
        type: string
        description: "Tenant ID."
        example: "1"
      application_name:
        type: string
        description: "Name of the application."
        example: "IIoT-App-1"
      description:
```

```
      type: string
      description: "Description of the application."
      example: "Internet of Things application"
    category:
      type: string
      description: "Category of the application."
      example:  "data-transfer"
    service_class:
      type: string
      description: "Service class of the application."
      example:  "high-throughput"
    importance:
      type: string
      description: "Importance of the application."
      enum: [LOW, MEDIUM, HIGH]
    rules:
      $ref: "#/components/schemas/APPIdentifRule"
    overlays:
      $ref: "#/components/schemas/AppIdentifOverlays"
    match:
      $ref: "#/components/schemas/APPIdentifMatch"
  required:
    - device_name
    - tenantid
    - application_name



AppIdentifOverlays:
  type: object
  properties:
    mode:
      type: string
      description: "Mode of Overlay path selection (static/dynamic)."
      enum: [static, dynamic]
    overlay_paths:
      type: array
      description: "List of static paths."
      items:
        type: string
      example: ["Overlay-100"]
    policy:
      type: string
      description: "Policy for dynamic paths."
    delay_threshold:
      type: number
      description: "Delay threshold for dynamic paths."
APPIdentifRule:
  type: object
  minProperties: 1
  properties:
    source_ip:
      type: string
      description: "Source IP address."
      example: ""
    destination_ip:
      type: string
```

```
      description: "Destination IP address."
      example: "10.168.77.53"
    protocol:
      type: string
      description: "Protocol."
      enum: [TCP, UDP]
      example: ""
    source_port:
      type: string
      description: "Source port."
      example: ""
    destination_port:
      type: string
      description: "Destination port."
      example: ""
APPIdentifMatch:
  type: object
  properties:
    match_name:
      type: string
      description: "Name of the match."
    match_attributes:
      type: array
      items:
        $ref: "#/components/schemas/APPIdentifMatchAttributes"
APPIdentifMatchAttributes:
  type: object
  properties:
    attribute_name:
      type: string
      description: "Name of the attribute."
    attribute_value:
      type: string
      description: "Value of the attribute."
OverlayReqName:
  type: object
  properties:
    tenantid:
      type: string
      default: "1"
    overlay_name:
      type: string
      example: "Overlay-100"
OverlayReqID:
  type: object
  properties:
    tenantid:
      type: string
      default: "1"
    overlayid:
      type: string
AppIdentifReqID:
  type: object
  properties:
    tenantid:
      type: string
      default: "1"
```

```
      app_identif_id:
        type: string
  AppIdentifReqName:
    type: object
    properties:
      tenantid:
        type: string
        default: "1"
      app_identif_name:
        type: string
  ServicePublicAccessParams:
    type: object
    properties:
      edge_name:
        type: string
        description: "Name of the edge device."
      tenantid:
        type: string
        default: "1"
      service_ip:
        type: string
      service_port:
        type: string
      service_protocol:
        type: string
        enum: [TCP, UDP]
  ServiceInterClusterAccessParams:
    type: object
    properties:
      overlay_id:
        type: string
      tenantid:
        type: string
        default: '1'
      service_ip:
        type: string
      service_port:
        type: string
      service_protocol:
        type: string
        enum: [TCP, UDP]
    required: ["overlay_id", "tenantid"]
```