SELF-ORGANIZED AND FULLY SERVICE-AUTOMATED MONITORING APPROACH AT THE CLOUD-NETWORK SLICE GRANULARITY

Kevin B. Costa[1,2], Felipe S. Dantas Silva[1,2], Douglas B. Maciel[1], Charles H. F. Santos[1,2], Augusto J. V. Neto[1,3], Fábio L. Verdi[4]
[1]Federal University of Rio Grande do Norte (UFRN), Brazil, [2]LaTARC Research Lab – Federal Institute of Education, Science, and Technology of Rio Grande do Norte (IFRN), Brazil, [3]Instituto de Telecomunicações, Brazil, [4]Federal University of São Carlos (UFSCar), Brazil

NOTE: Corresponding author: Kevin B. Costa, kevinbcosta@gmail.com

Abstract – *The cloud-network slicing concept has been established to deal with the advent of the Fifth Generation (5G) of mobile networks and its enabling technologies, thus promoting softwarization and cloudification. The Novel Enablers for Cloud Slicing (NECOS) ecosystem distinguishes itself over state of the art through the definition of slicing at both cloud and network levels and by promoting a Management and Orchestration (MANO) platform that provisions features with a self-organized and full-service automation approach across multiple federated domains. In the NECOS architecture, the Infrastructure and Monitoring Abstraction (IMA) component fetches Key Performance Indicators (KPIs) associated with the constituent parts of the active cloud-network slice instances managed by the NECOS platform. However, the design of the IMA monitoring component follows a centralized approach running at the core-cloud domain. Thus, the IMA concentrates on the monitoring data fetching function, done through interaction with measurement applications. It books all of them into a database and then forwards the incoming data for targeting management applications. Our findings in the slice monitoring state of the art study and assessments in IMA central cloud monitoring suggest that the centralized cloud approach cannot make distinct monitoring technologies compatible, besides not presenting a monitoring-as-a-service perspective that allows a self-organized and fully-service automated monitoring management scheme. In this regard, this work proposes the Distributed Infrastructure and Monitoring Abstraction (DIMA) multilevel monitoring plan. DIMA can promote monitoring as a service across an edge-cloud continuum inside the NECOS domain, enabling cloud/edge-centric and distributed monitoring schemes at the granularity of cloud-network slices' constituent parts. In order to assess the DIMA monitoring concept, we carried out a Proof of Concept (PoC) over a real-world testbed setup, confirming the proposed architecture's effectiveness and a practical view of the DIMA's impact in provisioning multilevel monitoring.*

Keywords – Cloud-network slicing, edge-to-cloud continuum, monitoring as a service, service automation

1. INTRODUCTION

The disruptive design of the Fifth Generation (5G) is based on a softwarized and cloudified ecosystem that aims to provide a flexible, dynamic, elastic, and scalable environment [1]. For example, Software-Defined Networking (SDN) [2] has shifted the paradigm from specialized hardware-based networks (such as switches and routers) to an ecosystem of software components running on clusters of general-purpose machines (cloud servers of data centers) [3]. Such software components have network resource management and programmability capabilities in a dynamic, granular, and scalable fraction from Network Function Virtualization (NFV) [4] enabling capabilities. The list of innovations in 5G networks includes service delivery in monolithic character, from isolated logical partitions that define different instances over the same available physical infrastructure, namely slices.

At the network infrastructure level, these partitions are known as network slices [5] and have become the norm among network operators in pursuing new business models. Technically speaking, a network slice is bound to virtual resources (e.g., containers or virtual machines, virtual network interfaces, bridges, virtualized network functions, and tunnels). In addition, the virtual resources are mapped onto physical network infrastructure resources (e.g., physical network interface with reserved bandwidth, servers), which are fully programmable to provide a monolithic end-to-end view [6].

The Novel Enablers for Cloud Slicing (NECOS) project [7] brings a more comprehensive slicing view by decoupling Management and Orchestration (MANO) features at the network and cloud levels. In addition, the project advances beyond the state of the art by designing a cloud-network slicing platform capable of operating across multiple domains while leveraging an architecture with dynamic end-to-end scope [8, 9] for a fully self-organized, service-automated, and template-driven MANO of services.

A cloud-network slice instance orchestrated by the NECOS platform is created from standardized templates, which abstracts the complexity of defining the use of the computational and network resources required for

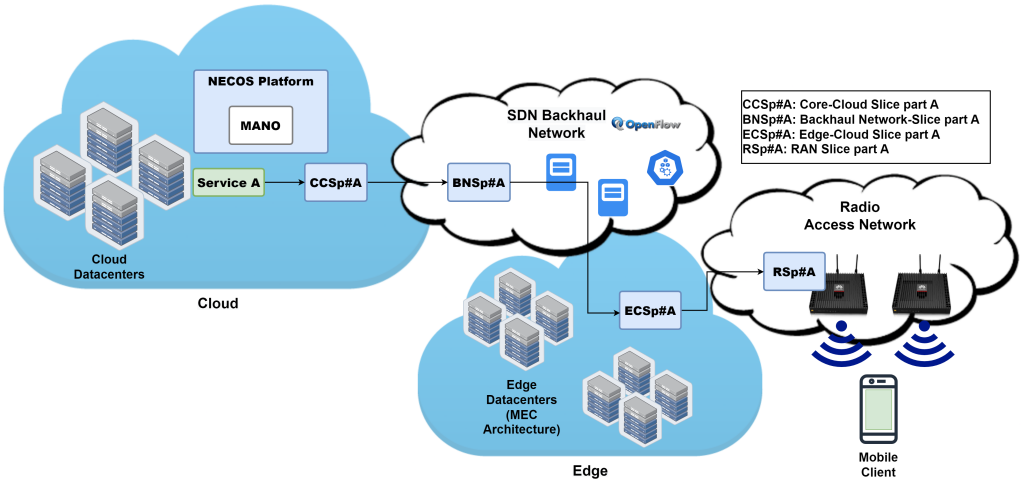

Fig. 1 – Hypothetical scenario for a cloud-network slice A

its deployment. NECOS templates represent the following groups of managed elements: (i) infrastructure resources (e.g., network, edge/cloud) and virtualized network functions; (ii) manageable resources (i.e., connectivity, computation, and storage); and (iii) application services, which features operations and attributes specially designed for the requirements of specific industry sectors.

In the self-organizing end-to-end cloud-network slicing service provided by the NECOS platform, each slice is composed of a set of constituent sub-units called slice parts [8]. The slice parts are mapped to physical and virtual infrastructure resources from the NECOS platform's MANO functions, with a wide range of both network- and cloud-level enabling technologies. Fig. 1 depicts a hypothetical cloud-network slicing deployment and its respective elements.

In the scenario described in Fig. 1, we consider a given deployed cloud-network slice *A* instance, which is composed of the following slice parts elements:

a) core-cloud slice part;

b) edge-cloud slice part;

c) network slice part backhaul: interconnects the core-cloud slice part and the edge-cloud slice part;

d) RAN slice part: interconnects the edge-cloud and RAN slice parts.

In the NECOS project, the MANO platform has as one of its functional requirements to provide monitoring of cloud-network slice instances, taking into account the granularity of constituent slice parts in a self-organized and template-driven manner. In addition, the NECOS platform has the functional requirement to continuously monitor all active slice parts during all their life cycles. Therefore, the Key Performance Indicators (KPIs) that can be monitored must be stored in a database accessible to architectural components responsible for maintaining the entire life cycle of the active cloud-network slices (e.g., elasticity control mechanisms).

1.1 Problem statement

Within the NECOS domain, determining which cloud-network slice parts are affected by system anomalies is a critical and challenging management problem. Analyzing the operational impact of monitoring and controlling at the finest granularity requires understanding: (i) the interdependency between the slice parts, (ii) the technologies involved in their orchestration and interfacing, and (iii) the impact of their use on the end-to-end cloud-network slice structure, among other several important aspects. In general, slice parts must be monitored continuously (with different reading periods) by collecting KPIs on their systems and technologies (e.g., communication and software interfaces).

In the NECOS ecosystem, the Infrastructure and Monitoring Abstraction (IMA) [10] component provides a northbound interface that allows the abstraction of the monitoring of slice parts. IMA is designed on a fully centralized approach, on components hosted in the network core-cloud. IMA creates management agents (called adaptors) for each orchestrated slice part, which runs in the central cloud as a Virtualized Network Function (VNF). Each adaptor performs KPI selection from a set of measurements available in the IMA database based on its criteria defined according to its functional requirements. To this end, applications provide monitoring KPIs by

delivering to IMA all possible collectible measurements on the monitored devices in the whole domain [11]. Upon receiving these, IMA books the entire information in a local database accessible through well-defined internal functions and interfaces in the NECOS [12] architecture.

Our findings in the related literature identified the employment of different monitoring modes in cloud/edge systems. The existing solutions are mainly based on deployments: (i) centralized in the core-cloud, (ii) partially distributed, and (iii) fully distributed, which adopt computing devices that are part of the whole interconnected system [13]. The discussion about the impacts of centralized versus distributed mode approaches goes back a long way. The study conducted in [14] suggests that distributed approaches can be adopted to minimize the data exchange rate between the involved systems without significantly reducing the overall system performance. Furthermore, distributed approaches can efficiently and scalably impact large-scale, highly dense systems [15].

Centralized monitoring levels, where a single system concentrates all data collection functions, are suitable for small systems. In these systems, the control is generally facilitated by requiring less effort to operate in a largely known environment. On the other hand, large-scale systems demand an approach with processing redundancy, aiming for more fault tolerance, capable of offering higher scalability rates and allowing a set of functions to be executed in predefined locations (e.g., central cloud or edge servers), these being natural properties of a distributed monitoring model. As the NECOS project assumes a large-scale system from a federation of different networks and cloud providers being defined by cloud-network slices, this research believes that the centralized monitoring approach of the IMA component needs to be revised to meet ultra-low latency rates required by 5G use cases.

By definition, the cloudified approach of the 5G ecosystem assumes that there are data centers distributed along the adjacent domain, with locations and capabilities distinct from each other and planned by the system operator. However, the European Telecommunications Standards Institute (ETSI) has defined the Multi-access Edge Computing (MEC) standard [16], which paves the way for extending cloud computing capabilities at the edge of the network as an integrated IT service environment. The motivation for this is that the MEC environment is characterized by ultra-low latency, high bandwidth, and real time access to information at the RAN level.

The MANO approach materialized in the NECOS platform fully adheres to the cloudified approach described above and is designed by the MEC paradigm. Furthermore, the NECOS platform enables a 5G physical infrastructure to provide customized and flexible networks through programmability and virtualization at client, application, service, and device granularity. In addition, the advancement of core-cloud capabilities in edge data center premises offers the expectation to improve operational performance, provide more agile services, leverage the high bandwidth available on end systems, and isolate security and privacy actions [17].

In this context, this paper proposes a self-organized solution for distributed monitoring of the constituent parts of cloud-network slice instances orchestrated by the NECOS platform along an edge-cloud continuum. In order to deliver the goals mentioned above, we propose Distributed Infrastructure and Monitoring Abstraction (DIMA). The functional architecture of DIMA has the premise of providing, in a fully automated way, multiple levels of monitoring to serve different infrastructures of the tenant.

1.2 Contributions

We summarize the main research contributions of this article as follows:

a) A comprehensive review of the state of the art solutions on network/cloud slicing monitoring scenario;

b) a monitoring as a service approach for enabling distributed measurements gathering at the granularity of cloud-network slice parts inside both cloud/edge domains;

c) a new distributed monitoring plan that integrates the NECOS platform for being capable of operating across an edge-cloud continuum;

d) a proof of concept deployment of the proposed DIMA monitoring plan in a physical testbed evaluation scenario.

1.3 Paper organization

The remainder of this paper is organized as follows. Section 2 synthesizes and provides introspection about a set of existing related works in the slice monitoring topic. Section 3 introduces the DIMA proposal in terms of functional requirements, functional architecture, and operation workflows. Section 4, in turn, devotes to providing assessment outcomes about the DIMA monitoring concept through a Proof of Concept atop a real testbed setting. Section 5 offers a discussion about the proof-of-concept assessments. Finally, but not least, Section 6 wraps up the paper by providing conclusions and future work suggestions.

2. RELATED WORK

Based on the solutions addressed in this review, the literature raises a few endeavors, specifically presenting study outcomes and introspection regarding existing slice monitoring proposals in the 5G network context. That said, this state of the art review initially aims to analyze proposals that act only in the cloud domain, followed by those

that act in the edge/RAN scope, and finally, existing proposals in the context of end-to-end cloud-network slices are considered.

At the end of all the related work analyses, we summarize the aspects expected for proposals of end-to-end cloud-network slices monitoring in the 5G context. Our analysis is based on the main limitations and contributions of the current state of the art and the considerations made in Section 1. Finally, Table 1 outlines the contributions of the present work against the state of the art.

2.1 Cloud-centric monitoring

In [18], a survey of the main network monitoring proposals for cloud scenarios is presented. The authors comprise monitoring tools such as Nagios[1], Zabbix[2] and IBM Tivoli[3]. In addition, it also reviews leading solutions for cloud monitoring, such as [19] [20], [21], [22], [23], [24], [25], and [26]. Although these proposals constitute the largest solutions used by the industry and incorporate the concepts of softwarization and virtualization of network components and functions, in addition to being limited to the cloud domain, they are not feasible for handling the scalability required by slicing scenarios since their KPI collecting entities are limited to a single instance of the collecting component. Moreover, they do not propose to bring the technology agnosticism expected in heterogeneous environments such as that of 5G networks.

In [27], the authors present a self-organized monitoring system for cloud slices. The proposed scheme can dynamically assign monitoring servers for extracting KPIs from cloud resources such as web servers, databases, and applications and implement algorithms for load-balancing the monitoring servers. Although this proposal foresees the performance in a slice context and proposes a technological abstraction for different infrastructure resources, its operation is limited to the cloud domain.

2.2 Edge and RAN-centric monitoring

In [28], an architecture of a Software Development Kit (SDK) called ElasticSDK is presented. The solution proposes to abstract the technological complexity of monitoring different RAN devices (e.g., 5G base stations) and to offer an abstraction of isolation complexity, security, and cost of monitoring data over multiple RAN technological domains in the context of slices. The system provides elastic (i.e., scalable and flexible) monitoring for applications that subscribe to its APIs. Furthermore, it offers filters that can be managed in real time by the subscribed given applications, acquiring only the data required for their operation.

[1]https://www.nagios.org/about/overview/
[2]https://www.zabbix.com/network_monitoring
[3]https://www.ibm.com/docs/en/tivoli-monitoring/6.3.0?topic=63-quick-start-guide

Although the motivation presented by this proposal is very similar to that of this work, the domain of action is limited to the RAN slice scenario. In addition, the architecture of the management component of the monitoring filters is centralized, so there are no adaptors capable of collecting, filtering, and delivering on-demand KPIs for each existing management application, creating bottlenecks as the central system needs to filter its data.

In [29], the authors propose a monitoring system to deal with the granularity of network-connected IoT devices in the edge domain. The solution focuses mainly on presenting a data model capable of standardizing and abstracting the monitoring complexity of various IoT devices. In addition, it creates a more generic message structure that should classify each KPI collected from each IoT device according to its characteristics (such as device type, value, source, state, and device description).

2.3 End-to-end cloud-network slices monitoring

The most recent proposals on monitoring in the context of end-to-end cloud-network slices in 5G networks remain in the view of generic architectures, with little focus on integrating innovative solutions in the context of more current MANO platforms, such as the NECOS platform. Moreover, the architectures analyzed mostly present a centralized approach to data provisioning, such as IMA. This is a negative point for an end-to-end cloud-network slices approach since concentrating the monitoring data in a central processing entity (usually located in the cloud) can impose latency in retrieving the monitored data, besides generating the need for data traffic that could be delivered locally. Consequently, a negative impact is expected on applications that could use distributed architectures to acquire such data as close as possible to the source (e.g., management applications located at the edge or even at RAN devices) for local network decisions, which only in exceptional cases should require communication with remote, geographically distant entities.

An example that accurately describes the scenario is the handover process in a multi-cloud context with multiple edge/RAN domains. In this example, a single Edge Data Center (EDC) could perform the handover process between multiple 5G Base Stations (5G-BSs) in the RAN. In this case, communication with remote entities can be avoided [30], as long as a user is transiting between the signals of 5G-BSs managed by the same EDC so that monitoring performed locally would enable lower decision latency by management applications, besides generating less traffic due to the constant RAN-Edge-Cloud communication. However, in the case of the transition to 5G-BSs from external domains, there is a need for an entity capable of contacting monitoring agents available in the other domains so that the handover process can occur accurately.

To meet this requirement, monitoring systems must be moldable, i.e., capable of orchestrating monitoring dynamics compatible with any requirements of applications that use MANO platforms (such as NECOS) to exercise their services in an end-to-end slicing context. To achieve this functionality, it is vital that the specialized monitoring subcomponents for each slice part (such as the adaptors in [11] and the MEFs in [31]) can establish direct interoperability with the applications that will consume the monitoring.

It is still necessary to point out that as presented in [32] and [33], end-to-end cloud-network slice monitoring is a functional requirement of MANO platforms. That said, monitoring plans in the slicing context must provide interfaces that allow MANO platforms, automated control of all mechanisms involved in the monitoring process (self-organizing plans), such as (i) network placement of the subcomponents responsible for collecting KPIs; (ii) access control across multiple network domains (e.g., federation scenarios, multi-cloud, etc.); and (iii) individualized configuration of each subcomponent responsible for monitoring each slice part (in view of allowing broad applicability for multiple monitoring requirements). Of course, such an approach imposes more data requirements on the monitoring plane APIs for full operation. However, adding these functional requirements does not negatively impact MANO platforms since they already have this requirement for implementing end-to-end cloud-network slices.

Another point that must be considered concerns analyzing the operational impact on MANO platforms. Our surveying of the related literature revealed that the concern regarding operational impact analysis is centered on evaluations regarding the effects of computational resource utilization (such as CPU and RAM usage). In addition, it demonstrates a deficiency in the performance impact analysis for MANO platforms, such as optimizing delivery time and network signaling generation, which directly impact the network regarding latency and available bandwidth.

In [31], a framework for monitoring 5G infrastructures consisting of multiple sites (i.e., multiple distributed infrastructures with long geographic distances) is presented. The proposal aims to make monitoring compatible with multiple technology domains through subcomponents called Metrics Extractor Function (MEF) that should collect and then send KPIs to a broker to submit this data for processing and delivery to consumers. Furthermore, the authors reinforce the concept of specialized components (such as adaptors) close to each type of device/VNF/monitoring agent (e.g., Netdata[4], Prometheus[5]) for collecting data from various types of technologies, as it enables the massive Machine Type Communication (mMTC) aspect of 5G, promoting technology agnosticism while maintaining flexibility in a distributed monitoring scheme.

In [11], the authors present and evaluate an architecture for a monitoring plan emphasizing elasticity capabilities, to be integrated into the NECOS platform. While the proposed architecture provides monitoring capabilities at the granularity of end-to-end slice parts, the architecture cannot handle the increased signaling generated across the network, with massive extraction of all monitorable KPIs sent to the cloud core. Furthermore, since the adaptors used for monitoring each slice part are aggregated to the centralized system, distributed applications and services that benefit from MEC approaches will negatively impact information collection time, which will need to travel the entire physical infrastructure to be collected in the cloud.

In [34], the authors propose a monitoring framework that implements edge and RAN distributed agents to collect KPIs from different monitoring solutions, such as FlexRAN and Prometheus. Centralized data availability to cloud-network slice consumers is the system's focus, either through dashboards or raw data availability through a centralized database. Although the solution has a distributed architecture capable of handling the technological granularity of the cloud-network slices and their slice parts, it remains in a centralized model for delivering KPIs. Furthermore, performance evaluations focus on using computational resources, with only one assessment focusing on the impact of KPI delivery time as the collection granularity (frequency of KPI collection) decreases.

[35] presents a framework called SliMANO, a plugin for end-to-end cloud-network slice orchestrators that promotes interoperability between MANO entities, SDN controllers, and RAN controllers in different network domains. The involved cloud-network slice monitoring does not present a detailed proposal for its operation. It only introduces a specification describing its functionality to notify the plugin in cases of slice failure to comply with the Quality of Service (QoS) standards determined by the tenant.

In [36], a framework called Lasagna is proposed. It is implemented as a module for end-to-end MANO network slices in WLANs following the IEEE 802.11 standard. The authors introduce the traffic rule feature, which allows end-to-end network slices to customize traffic policies for pre-allocated regions of the flow space of each wireless network device, thus reducing flow setup time and enabling support for network slicing in WLANs. In the monitoring scope, the system promotes the collection, filtering, and availability of KPIs such as the current number of frames in backlog, total airtime spent, number of packets and bytes transmitted, and number of packets and bytes dropped. However, in the monitoring context, the proposal cannot promote monitoring models capable of

[4]https://www.netdata.cloud/
[5]https://prometheus.io/

enabling an interplay between the wireless network domains and the network core or even allow a monitoring scheme from the RAN domain. Furthermore, the perspectives of impact analysis regarding the monitoring mechanism are not the focus of the evaluations since the KPI monitoring mechanism is separate from the proposal itself but from the MANO employed to implement the solution.

In [37], 5G-TRANSFORMER is presented. It proposes a MANO for end-to-end cloud-network slices focusing on autonomous resource allocation in mobile transport networks. Regarding monitoring, the MANO platform aims to provide APIs for tenant applications to monitor the network and consequently integrate, through the received data, the slice resources into their system. Furthermore, monitoring is proposed as a more abstract concept without presenting a functional architecture. However, the authors emphasize that for the end-to-end cloud-network slices scenario facing the demands of industry verticals such as automotive, e-health, media, and entertainment, the monitoring platform should be flexible to monitor the end-to-end infrastructure to meet the needs of such verticals.

Based on our state of the art analysis, we highlight four requirements that, according to discussions raised earlier and taking into account the 5G primary use cases (i.e., URLLC, mMTC, and eMBB), must be met for end-to-end slices monitoring to effectively promote an improvement of slicing management for MANO platforms in 5G scenarios.

- R1: distributed, aimed at optimizing the signaling cost of the monitoring process (emphasis on eMBB) and reducing the delivery time of KPIs (focus on URLLC) through local monitoring and delivery of KPIs in each slice part;
- R2: agnostic as to the technological heterogeneity of each slice (emphasis on mMTC);
- R3: customizable, to allow provisioning of multiple monitoring levels (centralized, distributed, and cloud-edge), being able to cater to different use cases of centralized or distributed applications and services;
- R4: self-organizing, from the perspective of allowing MANO frameworks (such as NECOS) to fully automate the orchestration of monitoring function settings across the domain, according to an indicated template service description.

Table 1 summarizes the main contributions and limitations of the current state of the art regarding the proposal of this work.

The study on related work indicates the need to develop a solution capable of offering a distributed monitoring service in the NECOS domain in the granularity of cloud-network slices. This paper fills this gap through the Distributed Infrastructure and Monitoring Abstraction (DIMA) proposal. The following section provides a full overview of the DIMA approach, presenting its architectural design and main components' operations.

Table 1 – Main contributions of state of the art against the contributions of this work.

Proposals	R1	R2	R3	R4
[31]	-	X	-	X
[11]	-	X	-	X
[34]	-	X	-	X
[35]	-	-	-	-
[36]	-	X	-	X
[37]	-	X	-	X
DIMA	X	X	X	X

3. DISTRIBUTED INFRASTRUCTURE AND MONITORING ABSTRACTION (DIMA)

In this section, we describe DIMA according to the elicited functional requirements, its architectural components, internal and external interfaces, and the flow of its operational procedures.

3.1 Functional requirements

The DIMA monitoring plan aims to cater to the following functional requirements:

1. It performs the gathering of the KPIs described in the template sent by the tenant at the granularity of each slice part instantiated by the NECOS platform. This will allow the provisioning of multilevel monitoring levels established along the edge-cloud continuum (i.e., fully cloud-centric, edge-centric, or hybrid). In this way, it is expected to settle a reduced signaling cost of the monitoring process (emphasis on eMBB) and reduced KPI delivery time (focus on URLLC) for applications whose functional requirements involve local-level decision-making across network domains.

2. It abstracts the different monitoring technologies used to collect KPIs associated with the different slice parts' constituents of a NECOS platform-commissioned cloud-network slice instance. Thus achieving agnosticism in the monitoring of slice part resources through a unified API, which should be consumed by the tenants' applications and services or by the MANO frameworks' mechanisms for decision-making that require KPIs related to the slice parts of end-to-end cloud-network slices.

3. It acts in an integrated manner with the NECOS platform to fully automate the orchestration of the monitoring functions throughout the domain, following the descriptions of received indications, aiming at a self-organized approach.

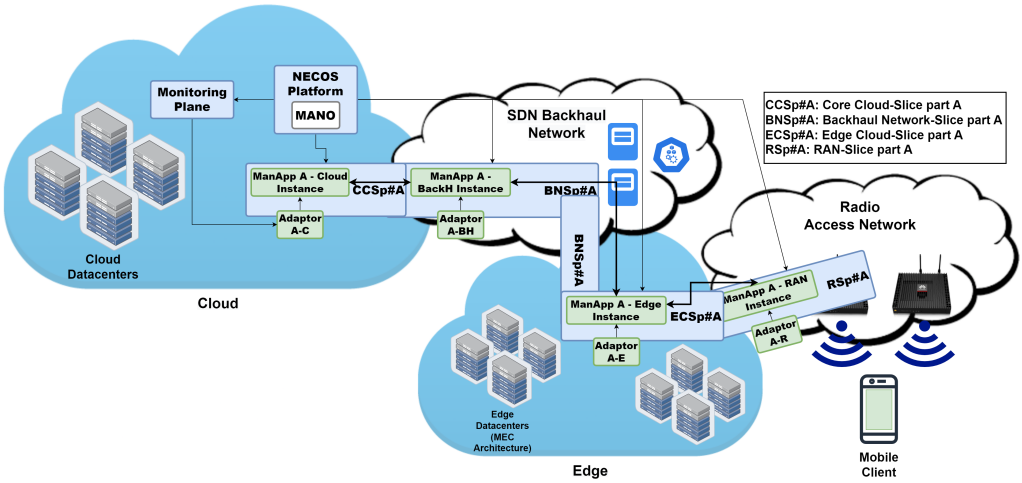

Fig. 2 – Illustration of the edge-to-cloud interplay monitoring level.

3.2 DIMA's functional architecture

The DIMA monitoring plan is based on the orchestration of isolated modules called Slice Part Adaptors (SPAs), which are moldable to the needs of the tenant according to the target monitoring resources and the available monitoring technologies (such as Netdata, Prometheus, among others) associated with each slice part. The procedure adopted by SPAs consists of collecting the KPIs described by the tenant and delivering them to the indicated management application. Additionally, depending on the available monitoring technology, filtering a set of measurements is necessary so that only those described in the template are sent (e.g., Netdata and Docker[6]).

As for the different monitoring levels available, the proposed architecture provides three levels capable of meeting the various use cases of the applications and management services. The following subsections explain each level and demonstrate possible use cases within the scope of tenants' applications and services and MANO platforms in the context of end-to-end network slices, which are directly impacted by the monitoring process and act in the new generation of mobile networks.

3.2.1 Edge-to-cloud interplay monitoring

This level involves implementing SPAs locally for each slice part of a given end-to-end slice to establish a network decision dynamics that occurs mainly locally in each domain through which a slice is instantiated. This level supports scenarios requiring ultra-low latency since the need to communicate with centralized and physically distant entities generates increased latency in the network decision process and involves signaling that interferes with the bandwidth available to the tenant in its slice.

As explained in Section 2, a handover process performed from the edge of the RAN itself clearly describes an example of this monitoring level's applicability. Fig. 2 illustrates the previously described scenario, where a slice *A* has a particular management application (e.g., called ManApp A) whose architecture is distributed, with multiple instances acting in various network domains. In order to enable interplay between such domains, isolated SPAs are implemented to collect, filter, and send specific KPIs to each instance of the management application.

Each slice part adaptor is designed and configured to meet the specific demands of a particular instance of the management application. For example, the A-C adaptor can be configured to collect KPIs that indicate CPU usage. In contrast, the A-BH adaptor is designed to collect only available bandwidth and memory consumption.

3.2.2 Full-cloud and full-edge monitoring

This level aims to monitor each slice part executed remotely from the cloud thoroughly. In this approach, SPAs must be deployed locally or remotely for each slice part. Still, KPIs must be collected and sent to be aggregated in the central entity monitoring plan and made available to the tenant or MANO framework in the cloud domain. This scenario should provide a global view of the end-to-end slice. An example use case for this approach concerns the end-to-end elasticity mechanisms of the NECOS platform, which need aggregated data from all the slice parts that constitute the cloud-network slice to properly perform the elasticity process (this level of monitoring is already largely provided in state of the art).

For the full-edge monitoring level, full monitoring of each slice part is performed entirely on the edge premises.

[6]https://www.docker.com/

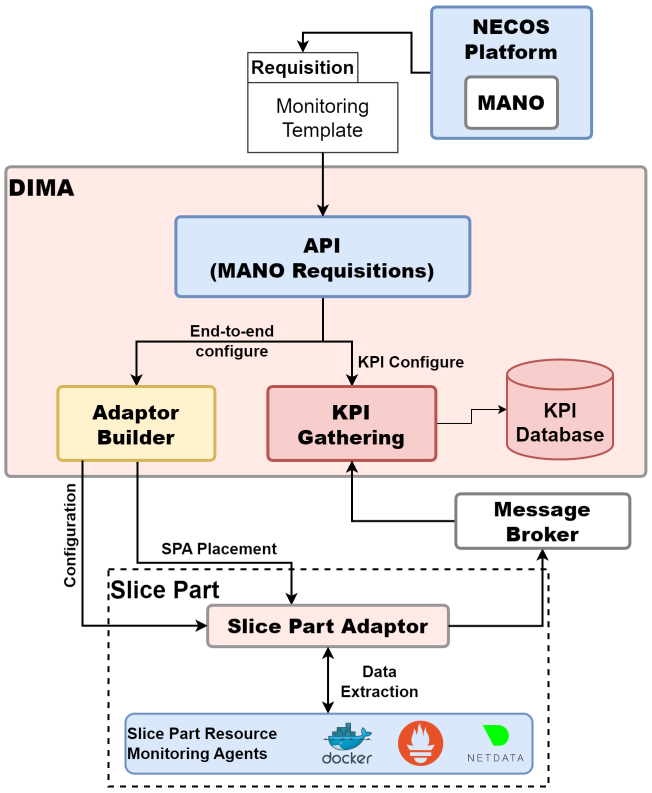

Fig. 3 – Main components and subcomponents of the proposed DIMA architecture.

Similar to the full-cloud monitoring level, it fits the profile of applications that act directly on edge premises. An example of the applicability of this level includes streaming data services, which require rapid adaptations given monitoring KPIs such as delay and packet loss to ensure a high-quality experience for end users. Another example that can be addressed is mobile online games that require high responsiveness because of monitoring throughput and jitter KPIs.

3.3 DIMA's components and subcomponents

The following subsections aim to describe in detail the functional components and subcomponents that form the architecture of the proposed monitoring plan. Fig. 3 provides an overview of the placement and relationship between the components and subcomponents of the proposed architecture.

3.3.1 API

The API subcomponent triggers all the processes performed in the DIMA monitoring plan. In addition, the API provides endpoints through which a MANO framework can request deployment, decommissioning, and updating the monitoring levels' instances. This requires the MANO framework to give a descriptor, called the monitoring template, containing information vital for implementing SPAs tasked with monitoring slice part KPIs. For example, Fig. 4 demonstrates a request made between the NECOS platform and the DIMA API for deploying an edge-to-cloud interplay monitoring layer.

```
{
'slice_id': 1, 'slice_name': 'slice-1', 'slice_parts': {
 'pcpe': [{
  'slice_part_name': 'wifi-slice-part','slice_part_id': 3,
  'pcpe_address': '10.7.227.130','monitoring_configuration':{
   'KPIs': {'KPI_1': '1 second', 'packet_loss': '2 second'},
   'monitoring_agent': 'node-exporter',
   'monitoring_location': [{'local': '10.7.227.130'},
  {'Access credentials': ['monitoring_user':'user_password']}],
   'delivery_location': [{'monitoring_delivery_place': 'local'},
   {'address': '127.0.0.1:7564'}]}}],
 'edge': [{
  'slice_part_name': 'edge-slice-part','slice_part_id': 2,
  'edge_address': '10.7.227.175','monitoring_configuration': {
   'KPIs': {'KPI_2': '1 second'},'monitoring_agent': 'Netdata',
   'monitoring_location': [{'local': '10.7.227.175'},
   {'Access credentials': ['monitoring_user':'user_password']}],
   'delivery_configuration': [{'monitoring_delivery_place': 'local'},
   {'address': '127.0.0.1:8775'}]}}],
 'cloud': [{
  'slice_part_id': 1,'slice_part_name': 'cloud-slice-part',
  'cloud_address': '10.7.229.191','monitoring_configuration': {
   'KPIs': {'KPI_3': '1 second', 'KPI_4': '1 second'},
   'monitoring_agent': 'prometheus',
   'monitoring_location': [{'local': '10.7.229.191'},
    {'Access credentials': ['monitoring_user':'user_password']}],
   'delivery_configuration': [{'monitoring_delivery_place': 'local'},
   {'address': 'localhost:9740'}]}}]}}
```

Fig. 4 – Monitoring template structure.

As presented in Fig. 4, the descriptor provided is structured on the premises of the distribution of the slice parts that form the end-to-end slice, where the placement of each SPA is based on the reference of its respective slice part. In this sense, it is necessary for the automated management of these SPAs by the MANO that they provide the same identifiers of the cloud-network slices and their slice parts internally assigned in the MANO framework during the end-to-end deployment process, as shown in the slice_id, slice_name, slice_part_id and slice_part_name fields.

Through these identifiers, the MANO framework will have access to remove the monitoring level assigned to the

indicated slice. For this, the API provides a specific endpoint for completely removing a monitoring level implemented for a given slice (receiving in the descriptor only the field 'slice_id').

Through the monitoring_configuration field, the API must receive instructions about: (i) the network placement for the deployment of SPAs, (ii) the KPIs that are to be monitored, (iii) the frequency (in seconds) with which they are to be collected, filtered and sent, (iv) the type of monitoring technology present on site (monitoring agent), and (v) the destination to which they are to be delivered. The pcpe_address, edge_address, and cloud_address fields describe the addresses where the slice parts are implemented in the resource providers' infrastructures, enabling SPAs to know where they should collect KPIs by querying the monitoring agents present.

3.3.2 Adaptor builder

The adaptor builder subcomponent is responsible for creating SPAs according to the instructions that a monitoring template (received by the API) carries and internally storing the descriptors of the monitoring levels currently implemented. To perform the placement, it has a set of mechanisms that perform remote access to the resource providers' infrastructures and data centers (using credentials sent through the monitoring template) so that once accessed, the SPAs placement is performed. The process of deploying SPAs consists of the following:

1. Query the information regarding the specific monitoring agent in the resource where the slice part is deployed. Once the technology of this agent is known, the adaptor builder retrieves (from its repository) a compatible version of the SPA to interact with the specific API of the specified monitoring agent, thus enabling the technological abstraction for the tenants' applications and services.
2. Run the mechanisms for remote access to the resources of each slice part described in the monitoring template.
3. Performs a configuration process on it, where it communicates to the SPA subcomponent, setup API, the previously received instructions about the KPIs it should collect through the monitoring agent, the collection frequency, and the delivery destination.

Finally, once the respective SPA is implemented and configured, the adaptor builder is also responsible for removing it. This decommissioning process requires a request to the API component describing the identifier of the slice. Once the API component requests the decommissioning, it will make an internal call to the adaptor builder component, which will internally query the monitoring level descriptor referring to the slice identifier provided to the API. The adaptor builder retrieves the address data of the resources where the SPAs are deployed, performs the remote access, and deactivates the respective running SPAs, requiring no request to the SPA API.

3.3.3 KPI gathering

KPI gathering acts as a KPI aggregator for slice part adaptors in the cloud. To store a large volume of KPIs over time, a database is designed to store and make the KPIs available. This approach aims to support applications and services running from the cloud that play roles that require a full view of a given end-to-end slice. Furthermore, applications and services that leverage Artificial Intelligence (AI) and Machine Learning (ML) to adopt proactive network decision models need KPIs monitored and aggregated over time to train the models correctly.

When implementing the requested monitoring level, if a slice part adaptor is located in the cloud, the adaptor builder system calls the KPI gathering, describing the identifiers of the slice and slice parts associated with the implemented monitoring level. These identifiers are sent to the KPI gathering subcomponent to configure queues in the message broker subcomponent to generate a manageable data traffic per slice part, where the KPIs collected by the SPAs will occur.

3.3.4 Message broker

The message broker organizes KPI traffic in a messaging model defined by queues. Each queue corresponds to an exclusive means of communication for KPI traffic from the given slice, where each KPI carries a header that identifies the slice part from which the KPI proceeds. This way, the KPI gathering component is capable of structuring in an organized way each KPI collected along the end-to-end slice and making them available in a well-defined path to the applications and services that will consume them.

3.3.5 Slice part adaptor

Fig. 5 depicts the SPA component, showing its subcomponents and roles. The SPA is a key tier of the DIMA architecture, for being responsible for interacting directly with the resources where the slice parts are implemented. Its role is to monitor, at a predefined frequency, KPIs specified through the monitoring template. It then abstracts the need for applications and services to know in detail about the necessary APIs of the various monitoring technologies made available by different resource providers, as in the scenario in federated networks of the NECOS platform.

SPAs are coded by mapping the API specifications of different monitoring technologies to the unified language of the DIMA API, generating a set of versions compatible with varying monitoring technologies. Then, according to the information about monitoring agents provided in the monitoring template, the adaptor builder can call an SPA

designed for a particular technology and enable technology abstraction.

As for the operation of its internal components, the adaptor builder instantiates the SPA. Once raised, it initializes its setup API subcomponent, where it waits for requests from the adaptor builder. Requests to the setup API guide its entire lifecycle and must contain: (i) the KPIs that are to be monitored; (ii) the collection frequency; (iii) the destination to which they are to be delivered; and (iv) the identifiers regarding the slice and slice part for which the respective SPA is deployed. Once this information is received and processed, the collector component is instantiated, thus receiving as instruction the KPIs that must be collected in the requests that the collector will make to the APIs of the monitoring agents, as well as the frequency with which these requests will occur. Finally, parallel to the instantiation of the internal collector component, the publisher component is instantiated. This subcomponent has the following functions: (i) format the data filtered by the collector, structuring a message with headers that indicate the slice and slice part of the monitored KPI; and (ii) send it to the destination specified by the request previously made to the setup API.

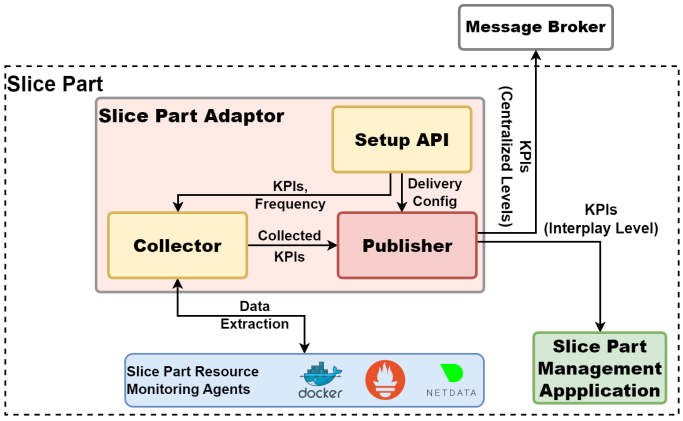

Fig. 5 – Subcomponents of the slice part adaptor.

3.4 DIMA's operation workflows

In order to better understand the practical operation of the DIMA platform, this section describes the main interactions between the various components. The explanations cover the processes triggered in deploying, removing, and updating monitoring levels for a given cloud-network slice and the processes performed for delivering KPIs to a given recipient.

3.4.1 Monitoring deployment for a cloud-network slice

The DIMA platform is guided based on the requests made by the MANO framework (in this work, the NECOS platform is used as a reference). Once the monitoring implementation is requested, a series of procedures are triggered to configure the internal components of DIMA and then perform the remote access to the providers' resources (selected in the NECOS marketplace [38]) for the instantiation of SPAs. Fig. 6 illustrates in more detail the operations performed in this process, presenting the flow of operations between the NECOS platform and the DIMA monitoring plan, along with the operations between the DIMA monitoring plan and the SPA subcomponent and how it delivers the collected KPIs to the requesting tenant application.

It is important to remember that the monitoring deployment and KPI delivery scenario is designed to encompass the delivery of KPIs at all three levels of monitoring supported by DIMA (i.e., edge-cloud interplay, full-cloud, and full-edge monitoring). The three monitoring levels differ only in the placement of the SPAs and how KPIs are delivered. At the edge-to-cloud interplay and full-edge levels, the delivery of KPIs to the tenant application is directly by the SPAs. In contrast, at the full-cloud level, the KPI gathering subcomponent located in the cloud performs the delivery. The operations described in Fig. 6 are explained below.

- 1.1 tenant.Request and 1.2 NSMANO.Response: The tenant requests MANO to deploy a monitored cloud-network slice. The NECOS platform will perform the necessary procedures for deploying the cloud-network slices. A confirmation message should only be returned to the tenant when the slice deployment is finished. Only after these procedures, the NECOS platform will contact DIMA to deploy the monitoring.
- 2.1 MANO.Request: Once the cloud-network slice is created, MANO will have all the necessary information to provide DIMA with the addresses, access credentials, and placement of the SPAs on the providers' resources. To enforce this information, MANO makes a request to DIMA through the SRO subcomponent.
- 3.1 DIMA.Request: The API subcomponent makes internal calls to the adaptor builder, KPI gathering, and broker subcomponents to configure them according to the specific end-to-end slice information and its slice parts that are to be monitored.
- 3.2 DIMA.Request: Once the adaptor builder, KPI gathering, and broker configuration process are complete, the adaptor builder starts the deployment process for each slice part provided in the monitoring template;
- 3.3 DIMA.Request: Once the SPAs are implemented, the configuration process is started. A request is made to each SPA's setup API providing information regarding which KPIs to monitor, the frequency of monitoring, and the delivery destination.
- 3.4 DIMA.SPA.Request and 3.5 DIMA.SPA.Request: Performs two internal calls to the collector and publisher subcomponents. Once the collector is configured, it will perform data collection from the API of

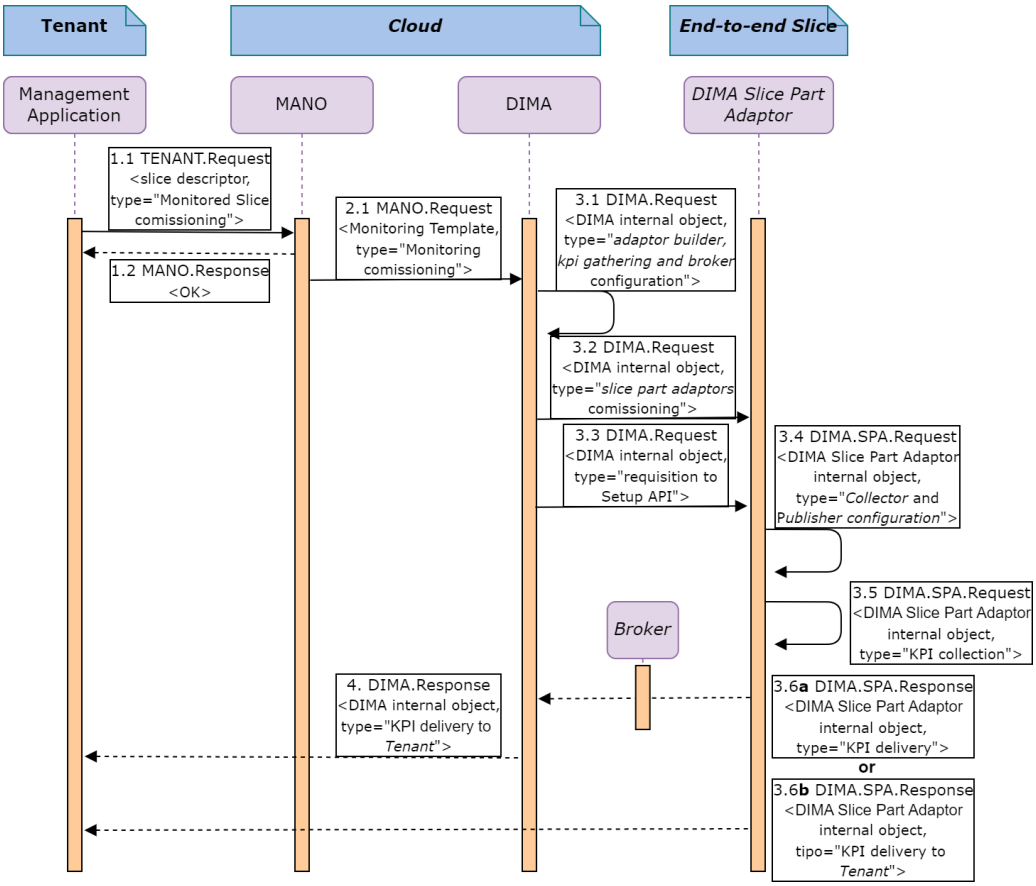

Fig. 6 – Main operations involved in the process of deploying monitoring of KPIs for a slice and the delivery to tenant applications.

the specified monitoring agent, perform data filtering to extract the requested KPI and then send it to the publisher. This process is achieved through a timer internal to the collector, which guarantees that the collection is performed according to the monitoring frequency established in 3.3 DIMA.Request.

- 3.6a DIMA.SPA.Request: This operation only occurs at the full-cloud monitoring level. After acquiring the requested KPIs, the collector sends the KPIs to the publisher. The SPAs will be configured to send the monitored KPIs directly to the tenant applications at the edge-to-cloud interplay and full-edge monitoring level. At the full-cloud monitoring level, the SPAs send the KPIs to the broker, making the collected KPIs available in a queue created specifically to traffic the KPIs of a given slice. Once available in the queue, the KPI gathering subcomponent retrieves this information from the queue, makes it available to tenant applications, and applies it to the KPI database. It is important to note that tenant applications need to implement mechanisms capable of receiving the KPIs sent by the SPAs, which will establish a network connection for the transmission of the collected KPIs.
- 4. DIMA.Response: If the requested monitoring level is configured as full-cloud if the KPI Gathering has retrieved the collected KPIs (operation 3.6a DIMA.SPA.Request), it delivers all the KPIs of each implemented SPA to the tenant application. Simultaneously it also performs the storage of all KPIs in the KPI Database.
- 3.6b DIMA.SPA.Request: If the chosen monitoring level is edge-to-cloud interplay or full-edge, the publisher sends the KPIs delivered by the collector directly to the tenant application.

3.4.2 Monitoring update for a cloud-network slice

The operations for updating monitoring instances involve requesting a monitoring template similar to the deployment from the DIMA API. If the identifier of the monitoring instance matches an operational instance, the update process is done by comparing the two templates. First, the descriptions of SPAs that do not appear in the updated template are deactivated. Next, the descriptions of slice part adaptors present in the template, which differ from the current instance in terms of KPIs collected, delivery destination, and monitoring frequency, will be removed and then redeployed with updated settings. Then, the descriptions of slice part adaptors provided in the updated template, which are not present in the current instance,

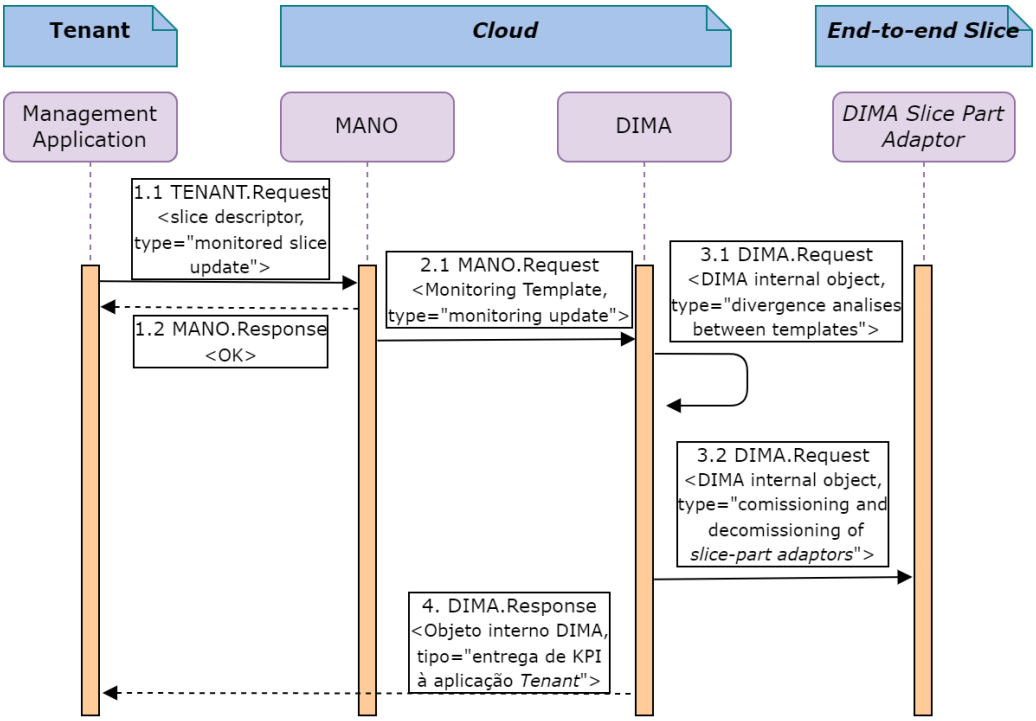

Fig. 7 – Main operations involved in the process of updating the monitoring of KPIs for a slice.

will be considered new slice part adaptors that should be deployed.

Fig. 7 illustrates in more detail the operations performed in this process, presenting the flow of operations between the MANO platform and the DIMA monitoring plan and the operations between the DIMA monitoring plan. Fig. 7 explains the procedures below.

- 1.1 tenant.Request and 1.2 NSMANO.Response: Tenant sends a request to MANO to update the monitoring instance.
- 2.1 NSMANO.Request: The MANO platform requests to the DIMA API, providing a new monitoring template based on the tenant descriptions.
- 3.1 DIMA.Request: The API subcomponent calls the adaptor builder subcomponent, which performs an internal check from the monitoring instance identifier provided in the monitoring template to retrieve the descriptor of the currently operational monitoring instance.
- 3.2 DIMA.Request: The adaptor builder triggers procedures for removing, deploying, or redeploying SPAs, according to the updated monitoring template.

3.4.3 Monitoring decommissioning for a cloud-network slice

Fig.8 illustrates the monitoring decommissioning process for a given slice.

- 1.1 tenant.Request and 1.2 MANO.Response: The monitoring decommissioning process consists of the tenant's request to the MANO platform. Once the MANO completes the slice decommissioning process, a confirmation is sent to the tenant.
- 2.1 MANO.Request: Upon the slice has been completely removed, the MANO requests the DIMA API monitoring plane to remove the monitoring from the slice. The request shall contain only the information regarding the slice that is to be removed, similarly to the one presented in Fig. 4, 'slice_id': 1.
- 3.1 DIMA.Request: Upon receiving the decommissioning request, the API subcomponent internally retrieves the instance of the adaptor builder subcomponent that has the information of the SPAs that are implemented for the slice.
- 3.2 DIMA.Request: The adaptor builder executes its remote access mechanisms to the providers' resources and finishes the instances of the running SPAs.
- 4.1 DIMA.Response: The API component sends an operation success confirmation message to the MANO platform.

4. PROOF OF CONCEPT OF THE DIMA PROPOSAL

This section presents a Proof of Concept (PoC) about the introduced DIMA monitoring plan. The evaluation's primary goal is to validate the proposed architecture, the

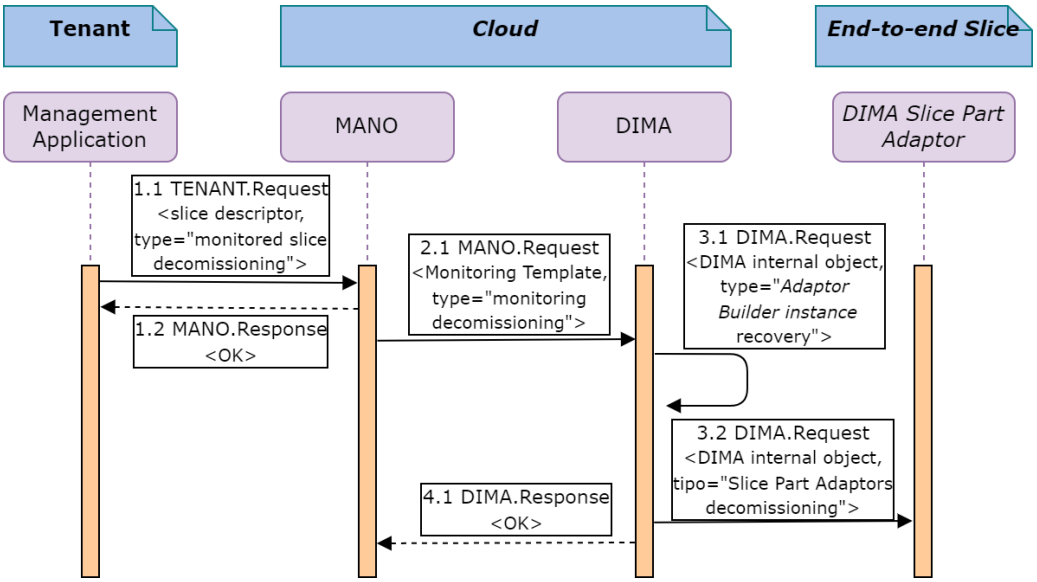

Fig. 8 – Main operations involved in the process of removing the monitoring of KPIs for a slice.

operation of the components as to workflows, and protocol interaction in its multiple levels of monitoring allowed. An instance of the DIMA prototype was implemented on a testbed available at the REGINA-Lab at the Federal University of Rio Grande do Norte, Natal, Brazil. The testbed comprises three main domains of resource providers, namely: (i) core-cloud, (ii) SDN network backhaul, and (iii) edge-cloud. The following subsections present the evaluation methodology, the DIMA prototype, the testbed, and usage scenarios. Then, we offer the achieved results with a discussion about DIMA's performance analysis and the lessons learned during the development and execution of the PoC.

4.1 Evaluation methodology

The evaluation methodology of this PoC consisted in verifying the viability of implementation and practical execution of the concepts presented in the previous section regarding the monitoring in the three levels idealized by the DIMA plan. To this end, DIMA is implemented to materialize the main components and subcomponents and the models of operations and exchanged messages.

The verification of the feasibility of the DIMA monitoring plan consisted of a practical exercise of its mechanisms implemented in a laboratory prototype to check the functional requirements presented in Section 3 were fulfilled. In this sense, three experiments were performed to reflect the three designed monitoring levels. The evaluation scenarios subsection presents in more detail the experiments for each monitoring level and their execution.

Each experiment is divided into three steps that must be successfully executed to validate the functional requirements fully. The steps are as follows:

1. Receipt of the monitoring template;
2. deployment of the SPAs according to the monitoring template specifications;
3. submission of the KPIs by the SPAs according to the destination provided.

These three steps are performed for the three defined monitoring levels.

For functional requirement #1, the KPI delivery time will not enter the scope of the DIMA prototype evaluation due to the testbed's limitation of very low network latency (evaluated at 4.8 milliseconds) between the different slice parts. Furthermore, the most likely scenario to consider the KPI delivery time would be in a testbed where the geographical distribution of the different domains of the federated network is on a kilometer scale, where the geographical distance would impact the traffic time of the measurements through the network, not being possible to adopt a testbed in these configurations due to a lack of resources.

After running the experiments, statistics related to resource utilization and the time cost of the developed prototype are presented for each monitoring level. This data raises a practical discussion regarding the feasibility and impact of the monitoring plan architecture proposed in this work. In this sense, insights are provided regarding the effect of different KPI delivery technologies on the slicing monitoring process, the specific implementation of the different components that constitute the monitoring solution used, as well as the different levels of monitoring experienced, given the particular circumstances of the laboratory testbed used.

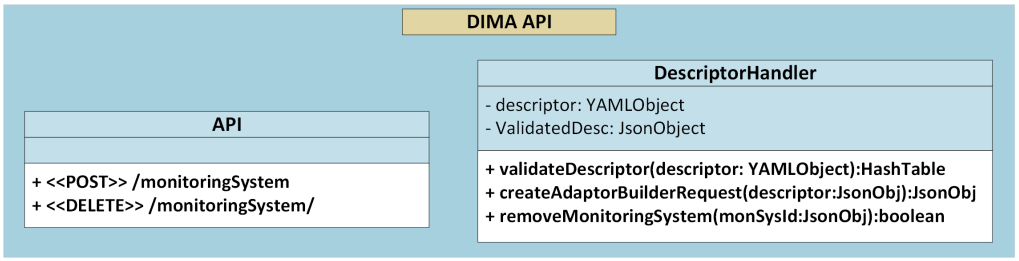

Fig. 9 – Class diagram of the core component API

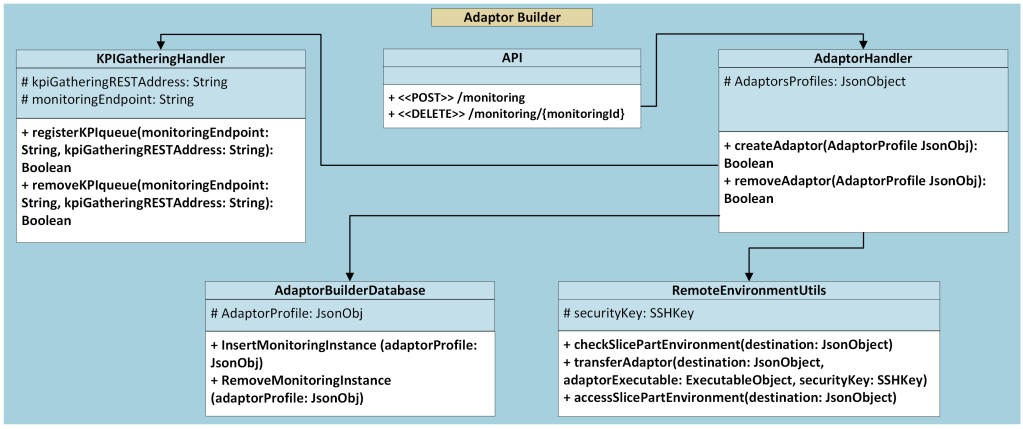

Fig. 10 – Class diagram of the core component adaptor builder.

4.2 DIMA prototyping

To validate the DIMA multilevel monitoring plan, a prototype was developed. The prototyping allowed a high degree of a learning curve, as well as validating the functional architecture regarding the efficiency of the proposed mechanisms, the practical feasibility of the mechanisms, defined interfaces, and communications. For simplification and the time cost of implementing the prototype, the evaluations covered only the monitoring deployment process, and the impact of the operation of the slice part adaptors once instantiated on the testbed.

The DIMA prototype was implemented based on a microservices architecture, where each component is independent in the execution of its routines and procedures between each other. The prototype was developed in Python 3 and Java 8 programming languages. This approach provided better perspectives of future improvements for implementing the DIMA architecture evaluated here. The following sections give an implementation overview of each DIMA component.

4.3 DIMA API

The class diagram of the DIMA API is shown in Fig. 9. First, the DIMA API establishes a request model for standardizing the monitoring template to be sent to DIMA. This template should contain the necessary information to trigger all the procedures required for implementing the monitoring. Once initialized, the DIMA API component instantiates two main objects: the API and *DescriptorHandler* classes. The API class is responsible for hosting two REST endpoints that drive the life cycle of DIMA. The REST endpoint of type POST, located in /monitoringSystem, receives the monitoring template (which must be structured in YAML format) for deploying monitoring instances of cloud-network slices.

4.4 Adaptor builder

The adaptor builder component hosts two endpoints for deployment (REST endpoint of type POST, located at /monitoring) and decommissioning of monitoring (REST endpoint of type DELETE, located at /monitoring, reporting the identifier of the monitoring instance). Fig. 10 shows the class diagram of the adaptor builder component. The *adaptorHandler* class triggers the functions for deploying (*createadaptor*) and removing (*removeadaptor*) SPAs. These functions use the *RemoteEnvironmentUtils* class to perform these procedures.

In the DIMA prototype, one of the main features needed for the adaptor builder was mechanisms for remote access to the environments where the MANO platform would provision slice parts. The adaptor builder can perform these tasks through the *RemoteEnvironmentUtils* class. Once it receives the remote access data related to the given slice part, this class can test the access and operation of the technologies required to run the slice part adaptor in the remote environment where it will be

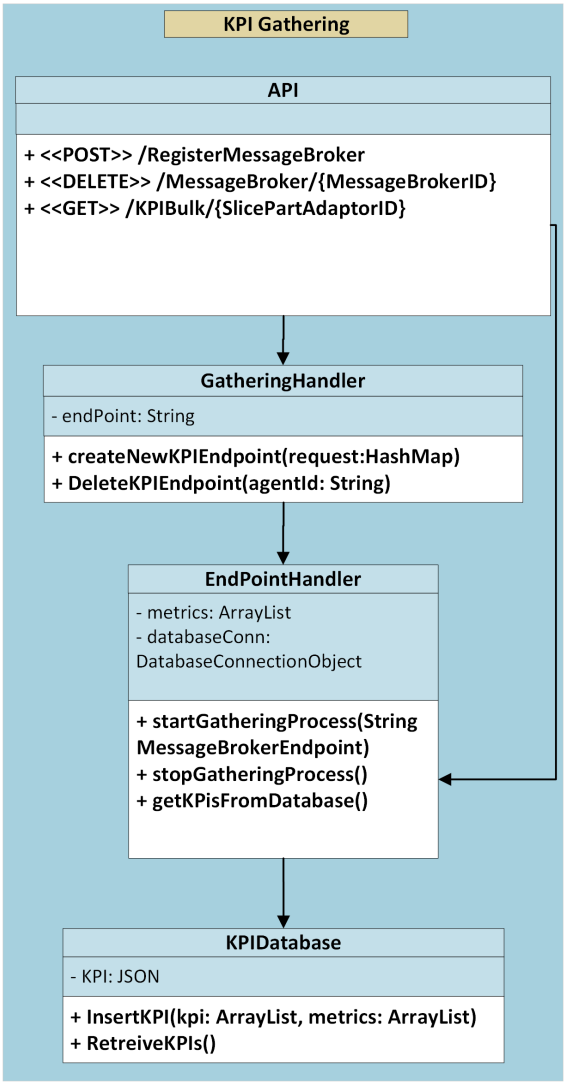

Fig. 11 – Class diagram of the core component KPI gathering.

deployed (*checkSlicePartEnvironment* function). Once access and the necessary technologies are properly operational, the class executes a function to transfer the version of the slice part adaptor indicated by the monitoring template over the network from an adaptor builder's storage directory (function *transferadaptor*).

With the procedures performed in the *adaptorHandler* and *RemoteEnvironmentUtils* classes resulting in the successful deployment of SPAs, the *adaptorHandler* class stores the descriptions regarding this SPA in a database. The functions that can interact with the database for this purpose are implemented in the *adaptorBuilderDatabase* class. In this class, the individual descriptions of each SPA are stored so that it will be possible to handle (in a future implementation) individual SPA change operations.

With the SPAs deployed and their descriptions stored in the database, the function *registerKPIqueue* from class *KPIGatheringHandler* is called by the *adaptorHandler* class if any of the SPAs are placed in the core-cloud domain.

4.5 KPI gathering

Fig.11 describes the class diagram of the KPI gathering component of DIMA. The KPI gathering must aggregate KPIs from SPAs placed in the cloud, making them available through a REST endpoint for each SPA that returns the set of KPIs monitored in the last 60 minutes. The KPI gathering connects to the message broker from the connection information received by the adaptor builder, in the *KPIGatheringHandler* class, in the *registerKPIqueue* function. The RabbitMQ[7] solution, a widely known messaging service in academia and work market solutions materializes the message broker, thus providing a queue system with high configurability.

4.6 Slice part adaptor

For the SPA, the main challenge is to make the different monitoring technologies compatible. The DIMA prototype achieves such compatibility by mapping the different endpoints of each technology used. The mapping occurs by identifying a communication pattern of the interfaces of several well-known monitoring technologies, such as Netdata, Prometheus, Docker Engine API, or even Zabbix[8]. It was identified through the study of these technologies that they all present interfaces for monitoring data through RESTful APIs. The class diagram of the SPA is shown in Fig. 12.

Through the comprehension of the REST pattern, the SPA was developed with the ability to map the different endpoints of each technology adopted in the evaluation scenarios. As an example, it assumes that Netdata is the technology capable of providing the CPU utilization of a given monitorable resource (such as virtualized resources in a Docker container), returning a history of 3.995 measurements from a request to the endpoint *api/v1/data?chart=<monitoring_target>.cpu*, where *<monitoring_target>* is the resource that can be monitored. For this specific case, it is necessary to check and filter the measurement set by removing the most recent measurement, check and standardize the unit of measure in which the measurements are provided, and finally format the KPI values and descriptions for sending to the specified target address.

Thus, for each KPI that is monitored by the DIMA prototype, there is a prior mapping of the set of procedures required to: (i) request the measurements from the technology and monitoring; (ii) process the received data, retrieving only the specific KPI or if it is a set of measurements of the same KPI, only the most recent one; and (iii) format the data within the standard KPI delivery default structure of the DIMA prototype. The standard KPI structuring for the implemented prototype can be seen in Fig. 13.

[7] https://www.rabbitmq.com/
[8] https://www.zabbix.com/

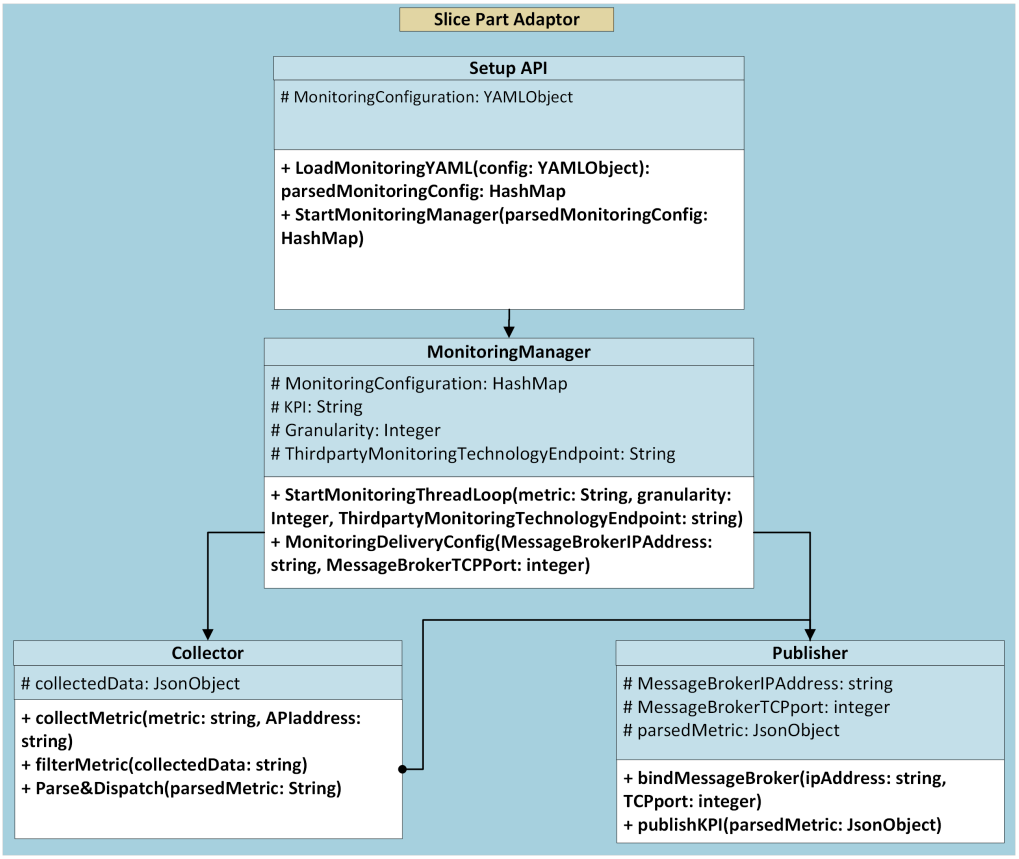

Fig. 12 – Class diagram of the slice part adaptor.

```
{
  'kpi': <kpi_name>,
  'value': <kpi_value>,
  'timestamp': <kpi_gathering_time>,
  'measurement_unit': <kpi_measurement_unit>,
  'sliceID': <kpi_slice_context>,
  'slicePartID': <kpi_slice_part_context>
}
```

Fig. 13 – DIMA KPI structure.

According to Fig. 13, timestamp, sliceID, and slicePartID provide context about the collected KPI, indicating respectively the exact moment of its gathering, the slice to which this measurement refers, and its associated slice part.

4.7 Testbed setup

Fig.14 presents the testbed used in the PoC. According to Fig. 14, the following elements compose the testbed:

- Core-cloud server: represented by a device featuring Intel Core i7-11800H 2.30GHz CPU with 16 cores, 32 GB DDR4 RAM, 750 GB SSD NVMe storage, with Docker VIM and Docker engine API monitoring technology.
- Edge-cloud server: represented by an HPCompaq 6005 Pro PC desktop (AMD Athlon II X2 B26 CPU, 16 GB DDR3 RAM, 250 GB storage capacity), with Docker VIM and Netdata monitoring technology.
- SDN backhaul network: represented by an HP Compaq 6005 Pro PC (CPU AMD Athlon II X2 B26, RAM 16 GB DDR3, 250 GB storage capacity), which emulates, through the OpenvSwitch[9], a virtual network bridge with two network interfaces to interconnect the core-cloud and edge-cloud domains.

The slice parts of the three domains are instantiated from the NECOS platform on the testbed. Both cloud and edge slice parts are containerized with virtualized compute and network resources using Docker VIM, responsible for

[9]https://www.openvswitch.org/

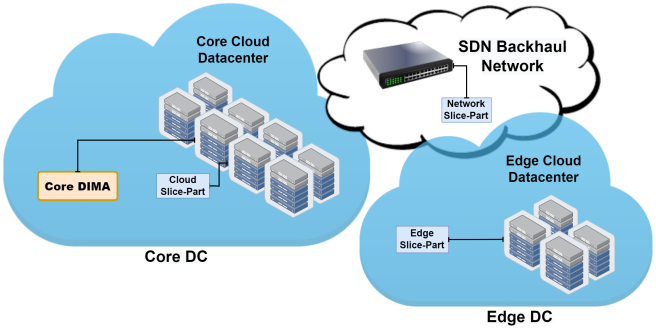

Fig. 14 – Overview of the testbed used in the PoC.

hosting a VNF for managing gNodeB (gNB) resources. In addition, the network slice part is provisioned with virtualized network resources through a virtual interface managed by OpenVSwitch, configured by the Ryu[10] WAN Infrastructure Manager (WIM) through the SouthBound OpenFlow API.

For validation of the plan-agnostic aspect of DIMA, the virtualized resources are monitored from their respective data collection technologies, with Netdata being the specific monitoring technology for the backhaul network and edge-cloud SDN domains and Docker engine API responsible for the core-cloud virtualized resources. With this scenario, a heterogeneous environment of monitoring technologies can be achieved.

Regarding the validation of DIMA to deploy different levels of monitoring, figures 15, 16, and 17 present the placement of the SPAs that DIMA deployed during the evaluation runs for each level.

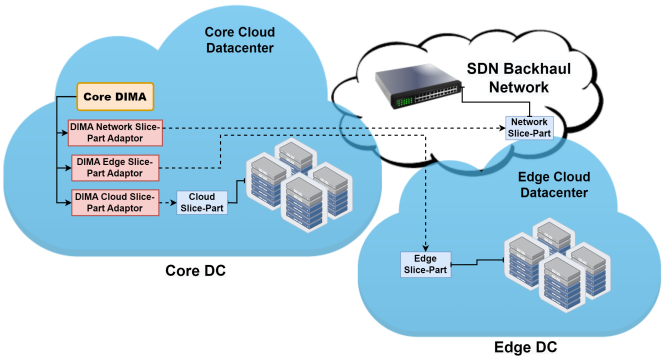

Fig. 15 – Placement of the slice part adaptors for the full-cloud monitoring level.

[10] https://github.com/faucetsdn/ryu

For the full-cloud monitoring level, the three SPAs are placed by DIMA in the core-cloud domain. Through this scenario the absence of traffic generated by the core-cloud SPA is expected, since its monitoring will be performed locally. In addition, the processing resources used in the core-cloud are far superior to the edge-cloud resources. Thus, it is also a scenario where reduced consumption of processing resources by SPAs is expected.

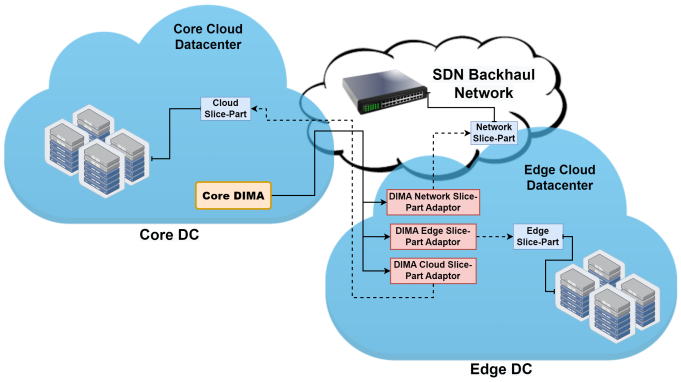

Fig. 16 – Placement of the slice part adaptors for the full-edge monitoring level.

In the full-edge monitoring level, the three SPAs are positioned by DIMA in the edge-cloud domain. Through this scenario the absence of traffic generated by the edge-cloud SPA is expected, since its monitoring will be local. In addition, the processing resources used in the edge-cloud are expected to be much lower than in the core-cloud. Thus, it is also a scenario where higher consumption of processing resources by SPAs is expected.

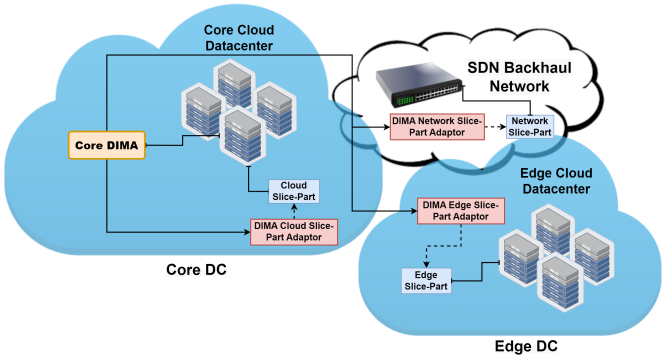

Fig. 17 – Placement of the slice part adaptors for the edge-to-cloud monitoring level.

Finally, the edge-to-cloud level presents the scenario where there is one SPA positioned in the core-cloud and responsible for monitoring the cloud slice part. Also, there is one SPA positioned in the edge-cloud and

responsible for monitoring the edge slice part, and one SPA positioned in the edge-cloud and responsible for monitoring the network slice part.

For all monitoring levels, the monitoring of the following KPIs, with their respective monitoring frequencies, is performed on each slice part of the testbed:

- Cloud slice part: CPU utilization (%); RAM memory utilization (%); traffic receive speed (Kbps/s); and traffic transmit speed (Kbps/s). All KPIs are collected at a frequency of 1 second.
- Network slice part: Network throughput (Kbps/s) with a monitoring frequency of 1 second.
- Edge slice part: CPU utilization (%); and RAM memory utilization (%). Both KPIs are collected at a frequency of 1 second.

From monitoring the KPIs collected by the SPAs, performance KPIs are collected regarding their CPU/memory usage, network throughput, and the number of network messages generated (which must be processed by the central DIMA component located in the core-cloud). These analyses will allow us to observe the impact the remote SPA deployment approach should have on the infrastructure. In addition, the cost of time to deploy each monitoring level is also analyzed to observe the cost involved in the remote deployment mechanisms of the adaptor builder component.

4.8 Analysis of results

This section presents the numerical results obtained in the experiments performed in each scenario of evaluating the multilevel monitoring plan in slice parts from the DIMA solution. In addition, the following subsections are intended to present the results collected from the behavior of the prototype in each experiment group. Finally, we provide a comparative discussion of the experimental results. It should be noted that the Proof of Concept is not intended to evaluate which scenario is the best but to prove that the solution can provide different monitoring levels.

4.8.1 Full-cloud scenario

As shown in Fig. 18, the computational costs of the SPAs at the full-cloud level were 9 seconds for the adaptor builder deployment of the three SPAs, 6.5% CPU consumption, and 5.1% RAM consumption. The CPU and RAM consumption measurements consider the joint cost of the three SPAs deployed.

The average deployment time of the SPAs was 3 seconds, considering that the adaptor builder, (located in the core-cloud) performed the deployment within the device where it was hosted. Furthermore, there is an average

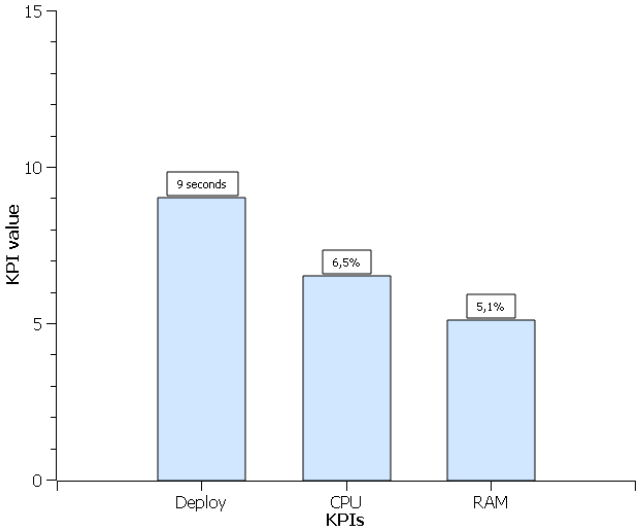

Fig. 18 – Time costs to deploy SPAs, CPU, and RAM usage for the full-cloud monitoring level.

consumption: of 2.16% CPU and 1.7% RAM per SPA. The average consumption of the SPAs was evenly distributed so that the variation in the number of KPIs collected by each does not impact the consumption of the computational resources evaluated (5 KPIs were collected from the core-cloud slice part and 1 KPI from the network slice part).

As for network costs, Fig. 19 shows that the joint throughput of the three SPAs in the core-cloud was 2304.82 Kbps and 382 messages per second. This means a network cost of only two SPAs, remotely monitoring the edge and network slice parts.

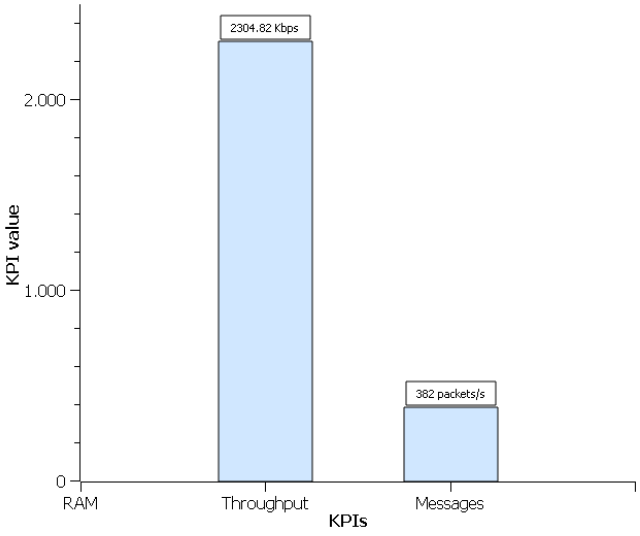

Fig. 19 – Network cost for full-cloud monitoring level

4.8.2 Full-edge scenario

Fig.20 presents the computational costs of the SPAs at the full-edge level. It took 37 seconds to deploy the three SPAs by adaptor builder, 17.4% CPU consumption, and 8.3%

RAM consumption.

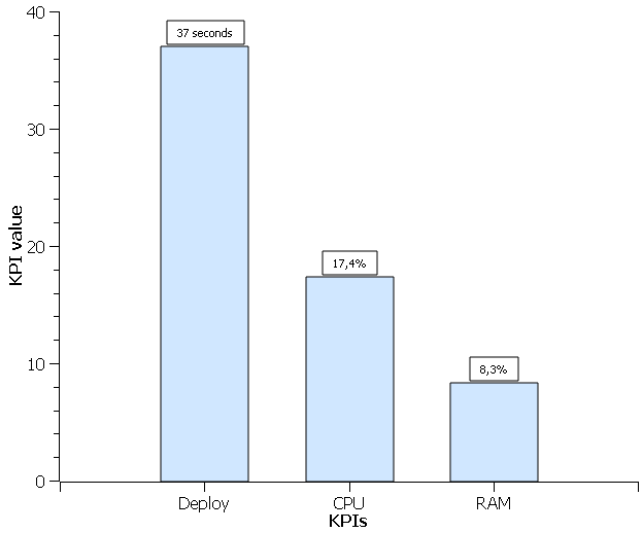

Fig. 20 – Time costs to deploy SPAs, CPU and RAM usage for the full-edge monitoring level.

The average deployment time for the SPAs was 12.3 seconds, considering that the adaptor builder (located in the core-cloud) performed the remote deployment of the three SPAs in the edge-cloud domain. In addition, there is an average consumption of 5.18% CPU and 2.76% RAM per SPA. Therefore, the average consumption of the SPAs was evenly distributed, just like in the full-cloud scenario.

As for the network cost, Fig. 21 shows that the aggregate throughput of the three SPAs in the edge-cloud was 306.28 Kbps and 95 messages per second. It means that this is a network cost of only two SPAs remotely monitoring the cloud and network slice parts.

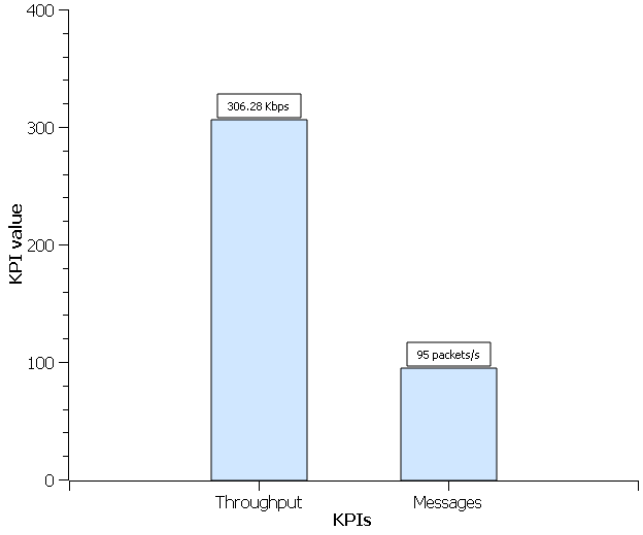

Fig. 21 – Network cost for the full-edge monitoring level.

4.8.3 Edge-to-cloud scenario

Fig. 22 indicates the impact of the computational costs of the SPAs at the edge-to-cloud level. It took 22 seconds to deploy the three SPAs by adaptor builder, 14.8% CPU consumption: (i) 1.6% CPU consumption by the SPA positioned in the core-cloud; and (ii) 13.2% CPU consumption by the two SPAs located in the edge-cloud domain. In addition, there was also 8.3% of RAM consumption being: (i) 1.7% of RAM by the slice part adaptor located in the core-cloud; and (ii) 5.5% by the two located in the edge-cloud.

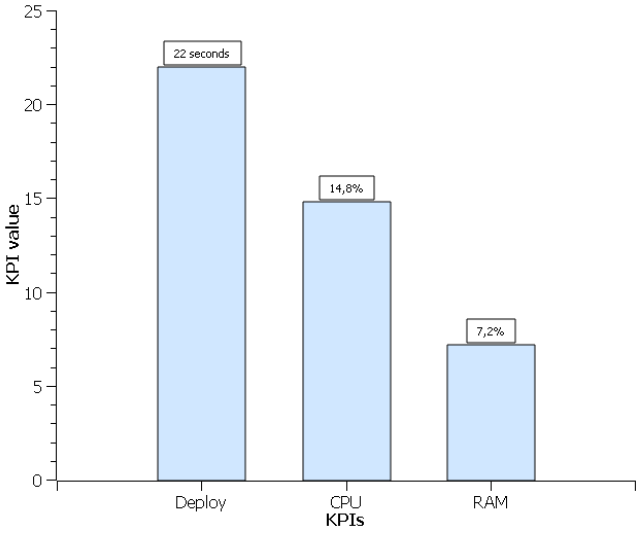

Fig. 22 – Time costs to deploy SPAs, CPU, and RAM usage for the edge-to-cloud monitoring level.

The SPA's average deployment time was calculated for each domain in which the SPAs were deployed.

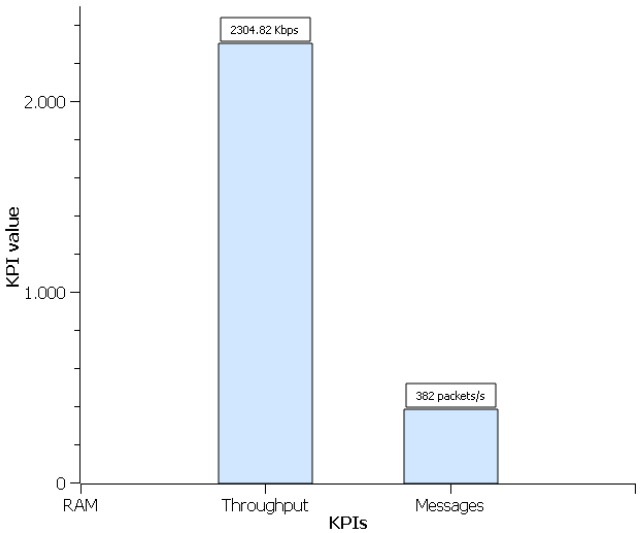

Fig. 23 – Network cost for the edge-to-cloud monitoring level.

As for network costs, Fig. 23 shows that the throughput obtained in the experiment was 198.64 Kbps and 71 messages per second. This means that this is the network cost

of only one SPA, remotely monitoring the network slice part since the two remaining were deployed to perform local-level monitoring.

5. OUTCOME DISCUSSIONS

This section aims to compile the individual results of each evaluated monitoring level, with the perspective of generating insights into the practical impacts of using the three monitoring levels.

5.1 Results comparison

Fig. 24 shows that the full-edge monitoring level had the longest time to deploy the SPAs. However, it can also be observed that the edge-to-cloud level had a lower time but about 244% longer than the full-cloud level. This behavior was observed due to the slice part adaptors transfer mechanism implemented in the adaptor builder. In the full-cloud scenario, it is not necessary to perform an SPA transfer procedure through the network external to the device hosting that domain, and the internal network interface of the device is used, which allows much higher throughput. Therefore, the results indicate that using mechanisms for distributed deployment of SPAs results in a slower deployment process that will consume the available bandwidth of the virtualized cloud-network slice network.

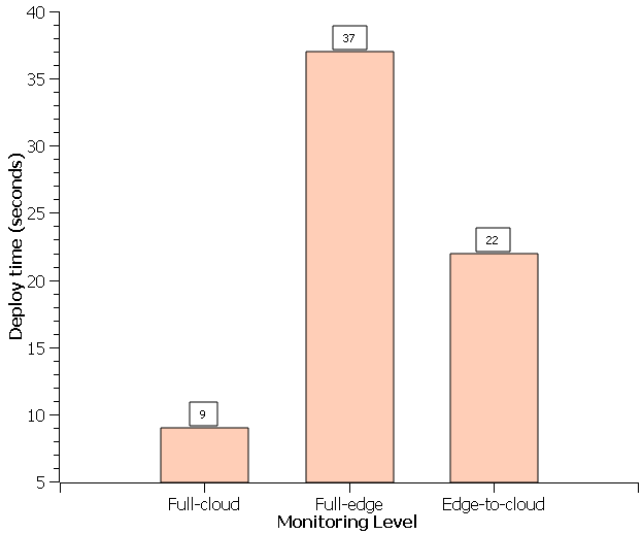

Fig. 24 – Time cost to deploy slice part adaptors at each monitoring level.

Figures 25 and 26 demonstrate that the full-cloud scenario had a throughput and network message processing cost inversely proportional to the deployment cost of the three tiers addressed, showing 1160.30% higher throughput and 538% more network messages generated than the edge-to-cloud scenario. Even though the SPA responsible for monitoring the cloud slice part was responsible for monitoring 4 KPIs. In comparison, the ones responsible for the network slice and edge slice parts together needed to monitor only 3 KPIs.

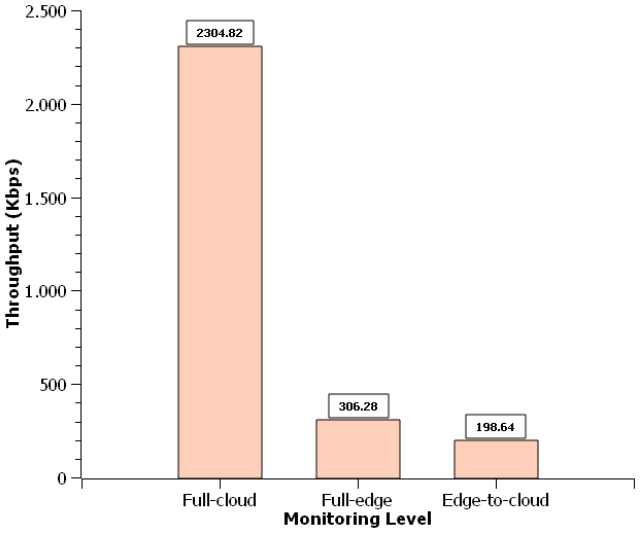

Fig. 25 – Network throughput generated by slice part adaptors at each monitoring level.

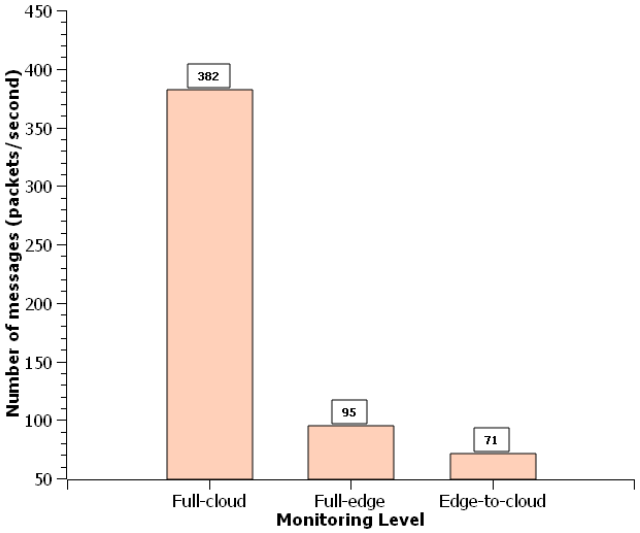

Fig. 26 – Sum of the number of network messages sent and received by the slice part adaptors at each monitoring level.

This behavior is explained by the specificities of the monitoring technologies adopted in the different domains. For example, the monitoring technology of the Docker engine API has a monitoring frequency limitation of 2 seconds between internal KPI measurements in its system and returns only the KPI that represents the value of the measurements at the exact moment they are requested. Netdata, on the other hand, can take measures at a frequency of 1 second, providing by default a history of measurements over time, resulting in a larger volume of data being received by the SPAs responsible for the edge slice part and network slice part monitoring.

In addition, the Docker engine API technology only covers monitoring resources virtualized by Docker VIM, unable to perform measurements of resources related to network slices (such as resources of OpenVSwitch, widely used for SDN network management). At the same time,

Netdata provides monitoring support for both.

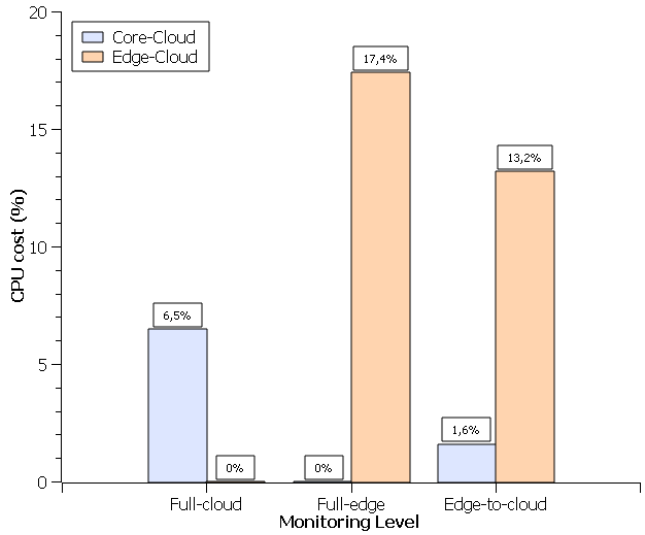

Fig. 27 – CPU cost of slice part adaptors in the core-cloud and edge-cloud domains at each monitoring level.

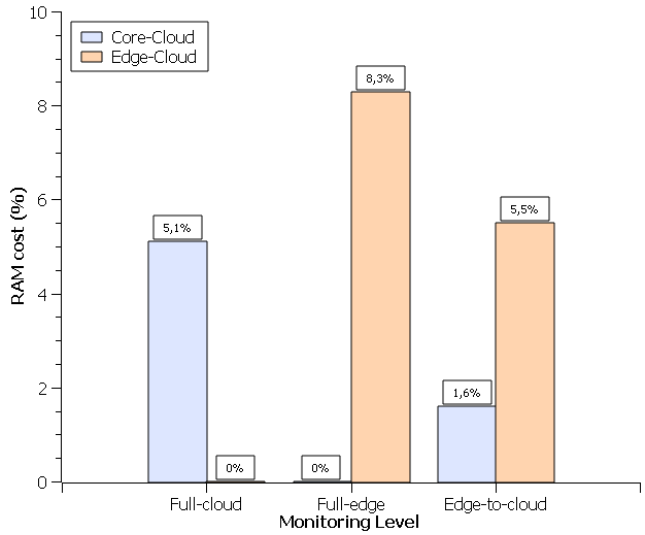

Fig. 28 – RAM cost of slice part adaptors in the core-cloud and edge-cloud domains at each monitoring level.

Figures 27 and 28 show that the CPU and RAM consumption for the full-edge monitoring level were 267.69% and 162.74% higher, respectively than the full-cloud level. First, however, it is necessary to consider that the resources of the core-cloud domain are higher in both CPU and RAM, with 16 CPU cores versus two cores (800% fewer cores) and 25 GB of RAM (amount of memory allocated to the core-cloud slice part) versus 16 GB of RAM (156.25% fewer RAM) of the edge-cloud server (amount of memory allocated to the edge-cloud slice part). In this sense, it is possible that in a scenario where the edge-cloud domain is provided with an edge data center, the computational burden of SPAs will be less significant.

5.2 Results conclusions

This section presents the perspectives for the Proof of Concept of the DIMA monitoring plan in the face of the achieved results.

Through the deployed DIMA prototype and the execution of experiments for each monitoring level, it was possible to verify that the implementation of DIMA could offer multiple monitoring levels besides the centralized ones in the cloud, thus validating the functional requirement #1. Furthermore, using the monitoring template and implementing SPAs allowed for abstracting the multiple monitoring technologies in the testbed.

In this sense, the functional requirement #2 was validated. Furthermore, it was possible to observe that the approach of multiple monitoring technologies in the testbed impacted the network cost (throughput and amount of messages generated) of the SPAs, demonstrating the viability of the edge-to-cloud scenario in a broader scope of use cases where it is necessary to isolate network decisions in specific domains, removing the signaling generated by SPAs positioned in the same domains where their corresponding slice parts are deployed.

It should also be noted that due to the computational resource limitations of the edge-cloud domain, the scalability of the SPAs in the realm of computational resource consumption when deployed in the edge-cloud was far inferior to the deployment in the core-cloud domain. However, deploying edge data centers with processing quantities comparable to cloud data centers is fundamental to using MEC architecture. Therefore, cloud and network providers are expected to possess more significant computational resources to cater to distributed architectures for cloud-network slice orchestration.

Finally, the results revealed that DIMA's self-organized model for deploying monitoring instances was validated, this being the functional requirement #3. Although a manual approach, i.e., requiring all procedures for instantiating multiple monitoring agents and requests between monitoring components to be manually initiated by a network operator, using a self-organized model still presents time disparities between the different levels evaluated.

One of the major impacts when evaluating the adopted levels is the time it takes to deploy SPAs. This is because the DIMA monitoring plan approach requires the transfer of SPAs across the available network. Hence, the more SPAs need to be deployed outside the domain where the core components of the DIMA architecture are positioned, the higher the time cost. However, it is important to note that the growth of deployment time cost per number of SPAs is linear.

6. CONCLUSION AND FUTURE WORK

This paper proposes the Distributed Infrastructure and Monitoring Abstraction (DIMA), a multilevel monitoring plan over edge-cloud domains orchestrated by the NECOS platform. The DIMA architecture monitoring plan differs from state of the art by enabling the orchestration of multilevel monitoring service along an edge-cloud continuum at the granularity of slice parts of active cloud-network slice instances.

We evaluate DIMA through a proof-of-concept operating in a real-world testbed. The assessment results offer diverse findings regarding the DIMA performance. First, DIMA transfers the slice part adaptors over the network, thus introducing a linearly increasing deployment time cost.

Furthermore, the experiments to verify the optimization of the KPI delivery time were not considered for evaluation to validate the DIMA concept. Still, they can be adopted for further performance evaluation. For this type of evaluation, a real federated network scenario is more advantageous, where the distance scale between different network domains is in kilometers, impacting a longer network traffic time between other domains.

In future work, we will evaluate the decommissioning of the SPAs from the adaptor builder into a new component capable of performing related procedures locally in each network domain by adopting REST requests. Also, we will experiment with DIMA over a geographically distributed cloud-network setup, thus validating the different monitoring levels under a more realistic scenario. Finally, it is of utmost importance to deploy DIMA facilities for monitoring RAN resources under several end-user mobile patterns, thus supporting handover decision-making procedures with real-time QoS KPIs.

REFERENCES

[1] 5G-PPP Software Network Working Group. *5G Vision: The 5G Infrastructure Public Private Partnership: the next generation of communication networks and services*. Tech. rep. 5G PPP, Aug. 2015.

[2] M. Boucadair and C. Jacquenet. *Software-Defined Networking: A Perspective from within a Service Provider Environment*. RFC 7149. Internet Engineering Task Force (IETF), Mar. 2014, pp. 1–20. URL: http://tools.ietf.org/html/rfc7149.

[3] João Batista Silva, Felipe S. Dantas Silva, Emídio P. Neto, Marcilio Lemos, and Augusto Neto. "Benchmarking of mainstream SDN controllers over open off-the-shelf software-switches". In: *Internet Technology Letters* 3.3 (2020), e152. DOI: https://doi.org/10.1002/itl2.152. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/itl2.152. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.152.

[4] European Telecommunications Standards Institute (ETSI). *Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV*. ETSI GS NFV 003 - V1.2.1. European Telecommunications Standards Institute (ETSI), Dec. 2014, pp. 1–13. URL: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf.

[5] Khizar Abbas, Talha Ahmed Khan, Muhammad Afaq, and Wang-Cheol Song. "Network Slice Lifecycle Management for 5G Mobile Networks: An Intent-Based Networking Approach". In: *IEEE Access* 9 (2021), pp. 80128–80146. DOI: 10.1109/ACCESS.2021.3084834.

[6] Xin Li, Chengcheng Guo, Lav Gupta, and Raj Jain. "Efficient and Secure 5G Core Network Slice Provisioning Based on VIKOR Approach". In: *IEEE Access* 7 (2019), pp. 150517–150529. DOI: 10.1109/ACCESS.2019.2947454.

[7] Felipe S Dantas Silva, Marcilio OO Lemos, Alisson Medeiros, Augusto Venâncio Neto, Rafael Pasquini, David Moura, Christian Rothenberg, Lefteris Mamatas, Sand Luz Correa, Kleber Vieira Cardoso, et al. "Necos project: Towards lightweight slicing of cloud federated infrastructures". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE. 2018, pp. 406–414.

[8] Stuart Clayman, Augusto Neto, Fábio Verdi, Sand Correa, Silvio Sampaio, Ilias Sakelariou, Lefteris Mamatas, Rafael Pasquini, Kleber Cardoso, Francesco Tusa, Christian Rothenberg, and Joan Serrat. "The NECOS Approach to End-to-End Cloud-Network Slicing as a Service". In: *IEEE Communications Magazine* 59.3 (2021), pp. 91–97. DOI: 10.1109/MCOM.001.2000702.

[9] Marcilio O. O. Lemos, Felipe S. Dantas Silva, and Augusto Venâncio Neto. "Analysis of Security Requirements in Federated Cloud Slicing Ecosystems: A Case Study of the NECOS Platform". In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. 2018, pp. 01271–01275. DOI: 10.1109/ISCC.2018.8538455.

[10] F. Verdi et al. *Intelligent Management and Orchestration*. Deliverable 5.2. Oct. 2019, pp. 1–70. URL: http://www.maps.upc.edu/public/D5.2%5C%20final.pdf.

[11] André Beltrami, Paulo D Maciel, Francesco Tusa, Celso Cesila, Christian Rothenberg, Rafael Pasquini, and Fábio L Verdi. "Design and Implementation of an Elastic Monitoring Architecture for Cloud Network Slices". In: *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2020, pp. 1–7.

[12] P. Papadimitriou et al. *Full API and Information Model Specification*. Deliverable 4.2. Aug. 2019, pp. 1–91. URL: http://www.maps.upc.edu/public/d4.2_final.pdf.

[13] Mark B. Moss. "Comparing Centralized and Distributed Approaches for Operational Impact Analysis in Enterprise Systems". In: *2007 IEEE International Conference on Granular Computing (GRC 2007)*. 2007, pp. 765–765. DOI: 10.1109/GrC.2007.130.

[14] Eduardo Cerqueira, Luis Veloso, Augusto Neto, Marília Curado, Paulo Mendes, and Edmundo Monteiro. "Q3M — QoS Architecture for Multi-User Mobile Multimedia Sessions in 4G Systems". In: MMNS '07. San Jose, CA: Springer-Verlag, 2008, pp. 38–49. ISBN: 9783540758686. DOI: 10.1007/978-3-540-75869-3_4. URL: http://doi.org/10.1007/978-3-540-75869-3_4.

[15] Raul Martinez Oviedo, Filipe Ramos, Sedat Gormus, Parag Kulkarni, and Mahesh Sooriyabandara. "A Comparison of Centralized and Distributed Monitoring Architectures in the Smart Grid". In: *IEEE Systems Journal* 7.4 (2013), pp. 832–844. DOI: 10.1109/JSYST.2013.2246033.

[16] European Telecommunications Standards Institute (ETSI). *Multi-access Edge Computing (MEC); Framework and Reference Architecture*. ETSI GS MEC 003 V2.2.1. European Telecommunications Standards Institute (ETSI), Dec. 2020, pp. 1–21. URL: http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.02.01_60/gs_MEC003v020201p.pdf.

[17] Davit Harutyunyan, Riccardo Fedrizzi, Nashid Shahriar, Raouf Boutaba, and Roberto Riggio. "Orchestrating End-to-end Slices in 5G Networks". In: *2019 15th International Conference on Network and Service Management (CNSM)*. 2019, pp. 1–9. DOI: 10.23919/CNSM46954.2019.9012732.

[18] Kaniz Fatema, Vincent C. Emeakaroha, Philip D. Healy, John P. Morrison, and Theo Lynn. "A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives". In: *Journal of Parallel and Distributed Computing* 74.10 (Oct. 2014), pp. 2918–2933. DOI: 10.1016/j.jpdc.2014.06.007. URL: https://doi.org/10.1016/j.jpdc.2014.06.007.

[19] Developer Guide. "Amazon CloudWatch". In: (2009).

[20] *Azure Monitor*. URL: https://azure.microsoft.com/pt-br/services/monitor/#overview.

[21] Aref Meddeb. "Internet QoS: Pieces of the puzzle". In: *IEEE Communications Magazine* 48.1 (2010), pp. 86–94.

[22] *Remote Management*. Aug. 2021. URL: https://www.teamviewer.com/pt-br/remote-management/.

[23] Bastian Hoßbach, Bernd Freisleben, and Bernhard Seeger. "Reaktives cloud monitoring mit complex event processing". In: *Datenbank-Spektrum* 12.1 (2012), pp. 33–42.

[24] Massimiliano Rak, Salvatore Venticinque, Gorka Echevarria, Gorka Esnal, et al. "Cloud application monitoring: The mosaic approach". In: *2011 IEEE Third International Conference on Cloud Computing Technology and Science*. IEEE. 2011, pp. 758–763.

[25] Vincent C. Emeakaroha, Tiago C. Ferreto, Marco. A. S. Netto, Ivona Brandic, and Cesar A. F. De Rose. "CASViD: Application Level Monitoring for SLA Violation Detection in Clouds". In: *2012 IEEE 36th Annual Computer Software and Applications Conference*. IEEE, July 2012. DOI: 10.1109/compsac.2012.68. URL: https://doi.org/10.1109/compsac.2012.68.

[26] Shirlei Aparecida De Chaves, Rafael Brundo Uriarte, and Carlos Becker Westphall. "Toward an architecture for monitoring private clouds". In: *IEEE Communications Magazine* 49.12 (2011), pp. 130–137.

[27] Márcio Barbosa de Carvalho, Rafael Pereira Esteves, Guilherme da Cunha Rodrigues, Clarissa Cassales Marquezan, Lisandro Zambenedetti Granville, and Liane Margarida Rockenbach Tarouco. "Efficient configuration of monitoring slices for cloud platform administrators". In: *2014 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2014, pp. 1–7.

[28] Xenofon Vasilakos, Berkay Koksal, Dwi Hartati Izaldi, Navid Nikaein, Robert Schmidt, Nasim Ferdosian, Riri Fitri Sari, and Ray-Guang Cheng. "ElasticSDK: A Monitoring Software Development Kit for enabling Data-driven Management and Control in 5G". In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Apr. 2020. DOI: 10.1109/noms47738.2020.9110373. URL: https://doi.org/10.1109/noms47738.2020.9110373.

[29] Binh Minh Nguyen, Huan Phan, Duong Quang Ha, and Giang Nguyen. "An Information-centric Approach for Slice Monitoring from Edge Devices to Clouds". In: *Procedia computer science* 130 (2018), pp. 326–335.

[30] Ming-Chin Chuang, Shuo-Ang Ke, and Chao-Lin Chen. "Network Controlled Handover Mechanisms in Mobile Edge Computing". In: *2020 International Conference on Information and Communication Technology Convergence (ICTC)*. 2020, pp. 756–761. DOI: 10.1109/ICTC49870.2020.9289161.

[31] Ramon Perez, Jaime Garcia-Reinoso, Aitor Zabala, Pablo Serrano, and Albert Banchs. "A Monitoring Framework for Multi-Site 5G Platforms". In: *2020 European Conference on Networks and Communications (EuCNC)*. IEEE, June 2020. DOI: 10.1109/eucnc48522.2020.9200914. URL: https://doi.org/10.1109/eucnc48522.2020.9200914.

[32] Alex Galis, Tusa Francesco, Stuart Clayman, Christian Esteve Rothenberg, and Serrat Joan. "Slicing 5G Networks: An Architectural Survey". In: May 2020, pp. 1–41. ISBN: 9781119471509. DOI: 10.1002/9781119471509.w5gref095.

[33] Douglas B. Maciel, Emidio P. Neto, Kevin B. Costa, Mathews P. Lima, Vitor G. Lopes, Augusto V. Neto, Felipe S. Dantas Silva, and Silvio C. Sampaio. "Cloud-Network Slicing MANO Towards an Efficient IoT-Cloud Continuum". In: *Journal of Grid Computing* 19.4 (Nov. 2021). DOI: 10.1007/s10723-021-09588-6. URL: https://doi.org/10.1007/s10723-021-09588-6.

[34] Mohamed Mekki, Sagar Arora, and Adlen Ksentini. "A Scalable Monitoring Framework for Network Slicing in 5G and Beyond Mobile Networks". In: *IEEE Transactions on Network and Service Management* 19.1 (Mar. 2022), pp. 413–423. DOI: 10.1109/tnsm.2021.3119433. URL: https://doi.org/10.1109/tnsm.2021.3119433.

[35] Flávio Meneses, Manuel Fernandes, Daniel Corujo, and Rui L. Aguiar. "SliMANO: An Expandable Framework for the Management and Orchestration of End-to-end Network Slices". In: *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*. 2019, pp. 1–6. DOI: 10.1109/CloudNet47604.2019.9064072.

[36] Estefanía Coronado, Roberto Riggio, José Villa1ón, and Antonio Garrido. "Lasagna: Programming Abstractions for End-to-End Slicing in Software-Defined WLANs". In: *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 2018, pp. 14–15. DOI: 10.1109/WoWMoM.2018.8449797.

[37] Antonio de la Oliva, Xi Li, Xavier Costa-Perez, Carlos J. Bernardos, Philippe Bertin, Paola Iovanna, Thomas Deiss, Josep Mangues, Alain Mourad, Claudio Casetti, Jose Enrique Gonzalez, and Arturo Azcorra. "5G-TRANSFORMER: Slicing and Orchestrating Transport Networks for Industry Verticals". In: *IEEE Communications Magazine* 56.8 (Aug. 2018), pp. 78–84. DOI: 10.1109/mcom.2018.1700990. URL: https://doi.org/10.1109/mcom.2018.1700990.

[38] Paulo D. Maciel, Fábio L. Verdi, Polychronis Valsamas, Ilias Sakellariou, Lefteris Mamatas, Sophia Petridou, Panagiotis Papadimitriou, David Moura, Asma Islam Swapna, Billy Pinheiro, and Stuart Clayman. "A Marketplace-based Approach to Cloud Network Slice Composition Across Multiple Domains". In: *2019 IEEE Conference on Network Softwarization (NetSoft)*. 2019, pp. 480–488. DOI: 10.1109/NETSOFT.2019.8806668.

AUTHORS

Kevin B. Costa graduated in computer networks from Instituto Federal do Rio Grande do Norte (IFRN). He received his master's in computer science from the Federal University of Rio Grande do Norte (UFRN). He is a researcher in the LaTARC Research Lab at IFRN and also a researcher in the Research Group On Future Internet Service and Applications Lab (REGINA) at UFRN. Currently, his areas of interest are cloud and network slicing, NFV, cloud and edge computing, network monitoring, and software development in Java and Python.

Felipe S. Dantas Silva is a professor at the Federal Institute of Education, Science, and Technology of Rio Grande do Norte (IFRN). He received his Ph.D. in computer science from the Federal University of Rio Grande do Norte (UFRN) in 2023. He is the Research Team Leader of the LaTARC Research Lab and his research interests include Future Internet, 5G/B5G, Software-Defined Networking (SDN), mobility control, cloud/fog computing, network/cloud slicing, machine learning, and security.

Douglas B. Maciel holds a master's degree in distributed systems and computing from the Federal University of Rio Grande do Norte (UFRN). He currently works as a researcher at UFRN and is a member of the REGINA Research Lab. His main research interests include Software-Defined Networks (SDNs), virtualization, network, and cloud computing, and network/cloud slicing. He is also interested in different programming languages and paradigms.

Charles H. F. Santos is currently pursuing an M.S. degree in computer science from the Federal University of Rio Grande do Norte. He is a lecturer at the Federal Institute of Rio Grande do Norte and a member of the LaTARC Research Lab. His main academic interests include vehicular networks and quality of experience in multimedia services.

Augusto J. Venâncio Neto is an associate professor at the Informatics and Applied Mathematics Department (DIMAp) of the Federal University of Rio Grande do Norte (UFRN), Brazil, working mainly in the field of computer networks and distributed systems. He is an IEEE senior member (with a Distinguished Lecturer in the period of 2024-2024), and a CNPq Research Productivity Fellow. He got his Ph.D. at the University of Coimbra (2008), Portugal, and is undertaking his 3rd post-doc at the University of Central Florida (2023-2024), USA. He is the author and co-author of more than 180 papers published in several conference proceedings and journals in the field of networking and telecommunications, regularly participates in conference organizing committees, and acts as a regular reviewer and technical program member as well. He has formed, as of now, 5 Ph.D. and 20 MSc students. He coordinates and participates in research projects afforded by national and international funding agencies. His current research interests fall in 5G/6G networks, slicing, Open-RAN, resource orchestration, cloud computing, and other fields.

Fábio L. Verdi is an associate professor in the Computer Science Department of the Federal University of São Carlos (UFSCar) Sorocaba campus. He has been working with data centers, cloud computing, SDN, and dataplane programmability, leading several national and international projects.