A Scalable Monitoring Framework for Network Slicing in 5G and Beyond Mobile Networks

Mohamed Mekki, Sagar Arora, and Adlen Ksentini, *Senior Member, IEEE*

***Abstract*—An efficient and scalable monitoring system is a critical component for any network to monitor and validate the functioning of the running services and the underlying infrastructure. This is more valid in 5G, as it relies on the network slicing concept, which adds many challenges to the monitoring system. Besides data isolation and multi-tenancy support, network slices require monitoring different types of resources, like RAN, computing, memory, network data rate, which belong to different technological domains, managed by different entities. Moreover, the monitoring system needs to be scalable, as in 5G a high-number of running network slices is envisioned. In this paper, we devise a novel monitoring framework for network slicing ready mobile networks, which features: 1) scalable monitoring system that supports a high number of running network slices in parallel; 2) technological domain agnostic thanks to a novel data collection (or monitoring) communication protocol; 3) support of multi-tenancy in a cloud-native environment. The framework has been implemented in a 5G facility, and its performance has been extensively evaluated.**

***Index Terms*—Monitoring systems, 5G, Network Slicing, Brokers.**

I. Introduction

MOBILE networks are seeing a significant shift in the last decade, with the development of a new generation (5G) and the next generation's foundation (6G). Although 5G release 17 specifications are not finalized, several commercial deployments are available using the NonStandalone (NSA) model, i.e., using 5G New Radio (NR) with 4G Core Network (CN). 5G introduces a radical evolution of the mobile network system. Besides improving the throughput and data rate, 5G introduces a new network architecture relying on the concept of network softwarization. On one hand, 5G NR provides larger bandwidth (up to 100 MHz in $<$ 6 GHz frequency band, and up to 400 MHz in $>$ 6 GHz frequency band) [1] to accommodate high data-rate demanding applications. Moreover, 5G NR introduces new physical layer numerologies that drastically reduce Radio Access Network (RAN) latency. Combined with Mobile Edge Computing (MEC) capabilities at the vicinity of the radio network [2], 5G NR will allow achieving a very low latency for services. On the other hand, 5G network architecture builds on the concept of network softwarization, which advocates for the usage of Software Defined Networking (SDN) and Network Function Virtualization (NFV) to build an agile CN and shed light on the concept of Network Slicing. The latter is a novel concept that aims at the partitioning of mobile network infrastructure into virtual network instances that are individually tuned to accommodate diverse services characterized by different requirements in terms of communication quality of service (QoS) within a common physical infrastructure. Network Slicing concept allows mobile operators to efficiently support the three envisioned classes of services using the same physical infrastructure: 1) enhanced mobile broadband (eMBB) for applications requiring high data rates, 2) massive machine-type communications (mMTC) intended to cover IoT applications that require support for a massive number devices, and 3) ultra-reliable and low-latency communication (uRLLC) for applications with strict requirements of communication latency and reliability.

A network slice is composed of sub-slices that use resources from different technological domains: Radio Access Network (RAN), CN, Edge/Cloud. The RAN sub-slice is constituted of Physical Network Function - PNF (Radio Remote Unit - RRU) and Virtual Network Function (VNF) (Central Unit - CU, Distributed Unit - DU), which are usually shared with other slices. The RAN sub-slice is also reserving enough Physical Resource Blocks (PRB) to guarantee its performances [3]. The CN sub-slice is composed of a set of VNF, which are shared with other network slices or dedicated to the network slice. Finally, the Cloud/Edge sub-slice is where the network service functions are run as VNF; all the VNFs are slice specific. The sub-slices are stitched together to build the end-to-end slice that runs a 5G network service.

Monitoring the performances (or Key Performance Indicators - KPI) of the running network slices is vital for both the network operator and the slice owner. For the former, measuring the KPI allows checking and troubleshooting the performances of the entities running the components of network slices, such as RRU, DU, CU, CN, Cloud/Edge, routers, etc; while for the latter, measuring KPI allows for checking the performance of the running services and validating the SLA signed with the network operator. It is obvious that the KPIs to measure for the network operator and for the slice owner are different. The slice owner is more interested in service level KPI [4] that corresponds to the service's performances such as the end-to-end latency, achieved throughput, consumed CPU, and memory of the VNF running the service, etc. While, the network operator is more interested in collecting KPI on the infrastructure components, such as radio resource usage, the radio latency, computing usage of shared VNF among slices,

Mohamed Mekki is with the Department of Communication Systems, EURECOM, 06410 Sophia Antipolis, France (e-mail: mohamed.mekki@eurecom.fr)

Sagar Arora is with the Department of Communication Systems, EURECOM, 06410 Sophia Antipolis, France (e-mail: sagar.arora@eurecom.fr)

Adlen Ksentini is with the Department of Communication Systems, EURECOM, 06410 Sophia Antipolis, France (e-mail: adlen.ksentini@eurecom.fr)

etc.

However, monitoring network slice KPI introduces several challenges. Among these challenges the fact that network slices use resources from different technological domains involving different entities based on different technologies. Indeed, the monitoring of RAN components is completely different from monitoring a NFV Infrastructure. Another challenge pertains to the scalability of the monitoring system, as it is expected that the network operator will run several parallel network slices on top of its 5G infrastructure. Finally, multi-tenancy and isolation among network slices need to be enforced; a slice related data should be seen only by its owner.

In this paper, we tackle these challenges by proposing a novel monitoring platform for 5G and beyond. The proposed framework natively supports network slicing and features a scalable architecture to handle a high number of running slices in parallel. To achieve this objective, we introduce metrics collectors deployed per network slice and follow the life cycle of the network slice (they are created when the network slice is deployed and are removed after the network slice is deleted). Besides, the proposed framework is technology agnostic when it comes to data collection, where a novel and technological agnostic monitoring protocol is introduced. Indeed, we propose a metric structure that unifies the management of the collected metrics and allows the association of the metric with the part of the network slice from which it was collected, hence linking between metrics from different domains. This will tackle the problem related to the heterogeneous monitoring system used by the technological domains where a network slice is running. Finally, the proposed monitor framework supports multi-tenancy and is cloud-native compliant. Indeed, besides supporting services that run in a cloud-native environment, all the framework components run as containers.

The paper is organized as follows. Section II presents related work and summarizes the main differences with our proposed framework. Section III presents and details the different components constituting our proposed framework. Section IV presents the results obtained when running the proposed framework on top of a 5G facility.

II. Related works

Several works addressed the challenge related to monitoring, especially in the context of cloud computing, which has been extensively studied in the literature. Different monitoring solutions have been designed to monitor traditional IT infrastructures and cloud environments, such as Prometheus[1]. In [5] a survey of cloud monitoring tools is conducted. It presents common characteristics, differences, strengths, and weaknesses of each reviewed monitoring tool. However, these solutions alone are not suitable for 5G relying on network slicing. Firstly, these solutions cover only one technological domain, the Cloud domain, while a network slice spans over different technological domains. Secondly, they cannot easily scale with the number of network slices, as only one component is in charge of collecting data, hence it constitutes the system bottleneck.

[1]https://prometheus.io/

As stated earlier, our objective is to provide a framework featuring an end-to-end monitoring solution for 5G supporting network slicing. To the authors' best knowledge, the proposed framework in this paper is the first end-to-end monitoring solution for 5G network relying on network slices, covering all the needed technological domains to deploy end-to-end network slices.

The work in [6] introduces a system that monitors multiple domains of a 5G infrastructure. The system consists of a metrics extraction function (MEF) that extracts and translates metrics, one MEF per monitored infrastructure component. The MEF extracts the metrics from the monitored component and exposes them to the upper layer where a broker system is deployed. A metrics aggregation component consumes the metrics provided by the MEFs from the broker and provides them to other tools responsible for metrics analysis and data visualization. Finally, a metric management entity is responsible for configuring the system entities. The system does not take into account network slicing and does not differentiate metrics from different slices. In addition, metrics are transferred without specific format or identifying elements, making it more suitable for one network slice at a time since deploying multiple services or slices will result in multiple unidentified data being stored in the system.

In [7] the authors introduce a prototype for RAN monitoring implemented on top of ElasticSearch[2] and FlexRAN [8]. It includes a producer API that writes measured data and statistics from the southbound control plane (using the FlexRAN controller) to the data store. The SDK has a filtering module that performs filtering operations such as selecting data or aggregating results. The solution focuses on RAN KPIs and can be considered a source of metrics for the RAN domain.

In [9], the authors propose an elastic monitoring solution for end-to-end cloud slices. The proposed system relies on other monitoring systems, called Monitoring Entities (MEs), and deploys adapters per ME and slice. The adapters collect the KPIs from the monitoring entities and send them to a distribution mechanism using RabbitMQ[3] to be then consumed by a slice aggregator that stores the KPI in a dedicated database. The slice identification is made at the adapter level, which creates a coupling between the management and deployment levels. In contrast, in our solution, the metric is mapped with the network slice part at the SO slice-specific collector using information coming from the sub-slice orchestrators (technological domain orchestrator), which allows keeping independence between the different entities involved in the monitoring process and the technological domain orchestrators. Last but not least, this solution does not collect RAN metrics.

In [10], the authors introduce a slice monitoring abstraction mechanism for data center slices relying on Lattice [11]. Monitoring a network slice resources is done via slice monitoring adapters using Lattice data sources, which are in charge of collecting the different KPI using probes. The latter collects relevant measurements for a segment of the end-to-end

[2]https://www.elastic.co/elasticsearch/

[3]https://www.rabbitmq.com/

slice. Besides disregarding the slice elasticity, the solution is designed for data center slices ignoring the RAN monitoring.

Work in [12] proposes a flexible monitoring framework that creates monitoring slices integrating cloud-specific and non-cloud monitoring solutions. Again, the authors did not discuss the scenarios of multi-clouds and multiple technological domains.

DASMO, introduced in [13], proposes the modification of the NFV architecture to support monitoring by embedding the monitoring elements into the Element Manager (EM) of a VNF. The work also tackles the monitoring's scalability, but no implementation nor performance evaluation of the solution have been conducted; it stays at the conceptual level.

Table I presents a comparison of our solution with [6], [9] and [10]. The three solutions share the same objective, which is monitoring platforms for network slicing.

III. A MONITORING FRAMEWORK FOR END-TO-END NETWORK SLICES

In this section, we will introduce the proposed monitoring framework of network slices in 5G and beyond networks. The section is divided into three parts: (1) the considered architecture for enabling monitoring in 5G; (2) the data collection servers and the monitoring protocol; (3) the data presentation.

A. Architecture

1) Network Slicing Management and Orchestration architecture: Before detailing the proposed monitoring framework architecture, we first introduce the network slicing architecture we build on. The latter is a generic architecture that allows orchestrating and managing the life-cycle of a network slice, including the monitoring process. It is based on the architecture introduced in [14] and shown in Fig. 1. The proposed architecture is composed of a SO, which is in charge of the Life Cycle Management (LCM) of network slices and monitoring its performance. The SO is equivalent to the 3GPP Network Slice Management Function (NSMF) [15] and exposes Northbound API (NBI) for the OSS/BSS or Communication Service Management Function (CSMF) as specified by 3GPP. The NBI covers the LCM of a network slice and the management of the monitoring process. The NS LCM API is already specified in [15] and manages the different steps of the network slice life-cycle, like commissioning, operation, and decommissioning. However, no API is specified to manage the monitoring of network slices'. Therefore, in this work we devised a new NBI to be exposed by SO to manage monitoring of network slices.

In the envisioned network slice architecture, each technological domain is managed and orchestrated by its own entity, known in 3GPP as Network Sub Slice Management Function (NSSMF). Depending on the technological domain, a NSSMF may correspond to NFV Orchestrator (NFVO) for Cloud/Edge domain, RAN Orchestrator (RANO) for RAN domain, and SDN controller for the case of the transport network domain. Examples of existing tools covering these functions are ONAP[4] and OSM[5] for NFVO, FlexRAN controller [8] for RANO, and OpenDayLight[6] and ONOS[7] for transport orchestrators.

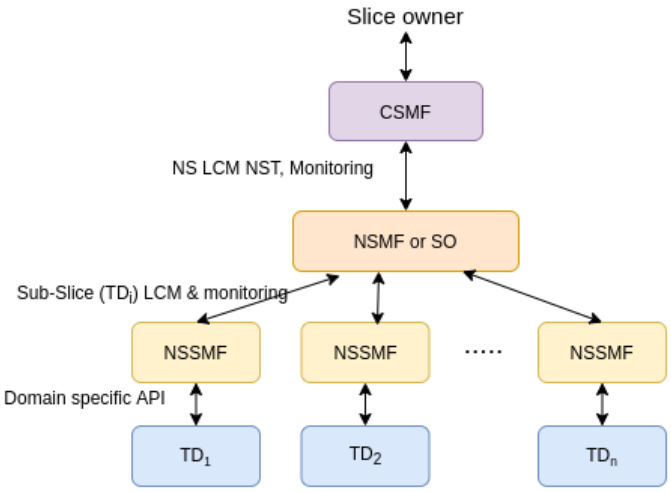

Fig. 1. Network slicing architecture

To deploy and instantiate a network slice, the CSMF or OSS uses the NBI exposed by SO to describe the needed resources (Compute and network) through a Network Slice Template (NST). It includes attributes and meta-data on the network slice (ex. the start date and end date, slice owner, type of slice, etc.), and information on each sub-slice composing the network slice. GSMA is providing examples of NST available in [16], namely Generic Slice Template (GST). To enable on-demand monitoring, we propose to extend the NST to indicate if monitoring is needed. If so, the slice owner should include the KPI list to monitor for each technological domain where the network slice is deployed. The template is created by the slice owner using an existing Blueprint provided by the operator or defining a new one through Intent. Each sub-slice (technological domain) composing the network slice is described in NST. For instance, in the case of the computing resource (i.e., Cloud/Edge domain), the NST may include information such as the number of CPUs, memory, and the virtualization technology (i.e., Virtual Machines - VM or containers) to be used. For the RAN domain, resources may be related to the functional split type [17], the MAC scheduler algorithm, the number of PRBs, and others. Finally, for the transport domain, resources may include the type of link (bandwidth, latency), number of Virtual Local Area network (VLAN), front haul link capacity, Virtual Private Network (VPN) links, and QoS. Each technological domain needed-resources are enclosed in the NST in the form of a technological domain-specific descriptor. For instance, for the Cloud/Edge domain, the resources are described using a Network Service Descriptor (NSD) that includes the VNF(s) list, the link to their VNF Descriptor (VNFD) or Application Descriptor (AppD) [18], how they should be interconnected, and the number of computing resources needed by each VNF. NST is passed by CSMF to NSMF when a network slice

[4]https://www.onap.org/
[5]https://osm.etsi.org/
[6]https://www.opendaylight.org/
[7]https://opennetworking.org/onos/

TABLE I
COMPARISON BETWEEN [6], [9], [10], AND OUR MONITORING SYSTEM

Aspect	[9]	[10]	[6]	Our system
Metrics source	From Monitoring Entities (Prometheus, NetData).	Lattice data sources.	Metrics generated by Infrastructure Components.	Kubernetes API, FlexRAN, plugins for Prometheus, Ceilometer.
Slice deployment	Support multiple slice parts on top of the same VIM.	Consider a VIM per slice.	Does not support network slicing.	Support multiple slice parts on top of the same VIM.
Monitoring management component	Engine Controller that communicates with the Slice Orchestrator.	Monitoring controller that communicates with the Slice Orchestrator.	Metrics Management Entity configures the system entities.	Monitoring engine integrated with the SO that Communicates with the NFVO and RANO.
Dynamic deployment	Use adapters per slice part.	Uniform on-demand monitoring layer per slice.	Metrics Extraction Function per infrastructure component.	Collection agent per slice part.
Slice elasticity	Horizontal elasticity is handled by deploying new adapters and monitoring entities. Vertical elasticity is handled by default.	Does not consider slice elasticity techniques.	Introduction of a new infrastructure component is by deploying a new Metrics Extraction Function.	Slice horizontal elasticity and the integration of new technological domains is handled by the deployment of new metrics collectors. Vertical elasticity is handled by default.
Metrics exposition	Expose the metrics to the Slice Orchestrator.	The metrics can be consumed by the Slice Orchestrator as feedback for the execution of its services and functions.	Expose metrics to data visualisation and analysis tools.	Expose Metrics using RabbitMQ and Grafana to the slice tenant.
Technological domains	Cloud, with support of multiple clouds.	Cloud, with support of multiple clouds.	Cloud and RAN.	Cloud with support of multiple clouds, and RAN with support of the FlexRAN controller.
Metrics abstraction	provide the element ID with the metrics.	No abstraction.	No abstraction.	Introduces a data collection protocol for multi-domain network slices.

creation is requested.

2) The proposed monitoring framework architecture: Building on the network slicing architecture (Fig. 1), we introduce in Fig. 2 the overall architecture of the proposed monitoring framework, where two technological domains (RAN and Edge/Cloud) are considered and detailed. Besides the management and orchestration entities mentioned earlier (i.e., SO, NFVO and RANO), the proposed architecture includes Data Collection Servers, a Data Presentation Server, and network slice specific components. The data collection servers are composed of Brokers deployed at two different levels: a high-level Broker is used to expose collected monitoring data to the end-users in RAW format, and a low-level Broker to collect data from the different technological domains. The high-level Broker is based on RabbitMQ, while the low-level Broker is relying on Kafka. In one of the next sections, we will explain in more details the motivation behind this choice. On the other hand, the presentation server is in charge of displaying the collected data per slice to the slice owners via a dashboard and graphical user interfaces. The presentation server consumes the monitored data per network slice stored in a Database (DB) at the SO, known as KPI DB.

It should be noted that the proposed framework's critical components are the slice collection agents that collect monitoring data of each sub-slice and aggregate them per slice. These agents are instantiated when the network slice is created and active throughout its life-cycle. In the considered scenario of two technological domains, three agents are instantiated per network slice, one by the SO and two by the technological domain orchestrators, i.e., at the RANO and NFVO. The role of the SO-level agent is to consume the data published by the slice specific agents, add meta-data on the network slice, and push it to the high-level Broker. After that, the data will be consumed by (i) the slice owner as RAW data; (ii) the data presentation server to be displayed via the dashboard. Whereas the role of the agents instantiated by the technological domain orchestrators is to collect data from the infrastructure, format the obtained data by adding meta-data, and push it to the lower-level Broker of the data collection server, i.e., Kafka.

Once the network slice is terminated, these three elements are deleted. Note that we assume that the slice agents are instantiated as containers and run in the Cloud/Edge domain. Obviously, using slice specific agents will guarantee that the proposed monitoring framework scales with the number of slices. Besides, if another technological domain is used, such as the transport network domain, only another collector agent is added; without impacting the overall system and the other agents' functions.

3) Data Collection procedures: As mentioned in the precedent section, the data collection is done by the slice specific collectors, which are instantiated by the different entities involved in the network slice LCM, i.e., SO, and technological domain orchestrators when the network slice is created. In this section, we will detail the instantiation process of the slice specific collectors and their functions, considering two technological domains RAN and Cloud/Edge, managed by RANO and NFVO, respectively.

a) SO: SO is the entry point of the system. It interfaces with the slice owner via OSS/BSS or via CSMF. It receives requests to deploy a network slice in the form of a NST. To recall, the NST includes details and attributes on the network slice to deploy, including the list of KPI to measure for each technological domain. The SO relies on the API exposed by the NSSMF (in our case RANO and NFVO)

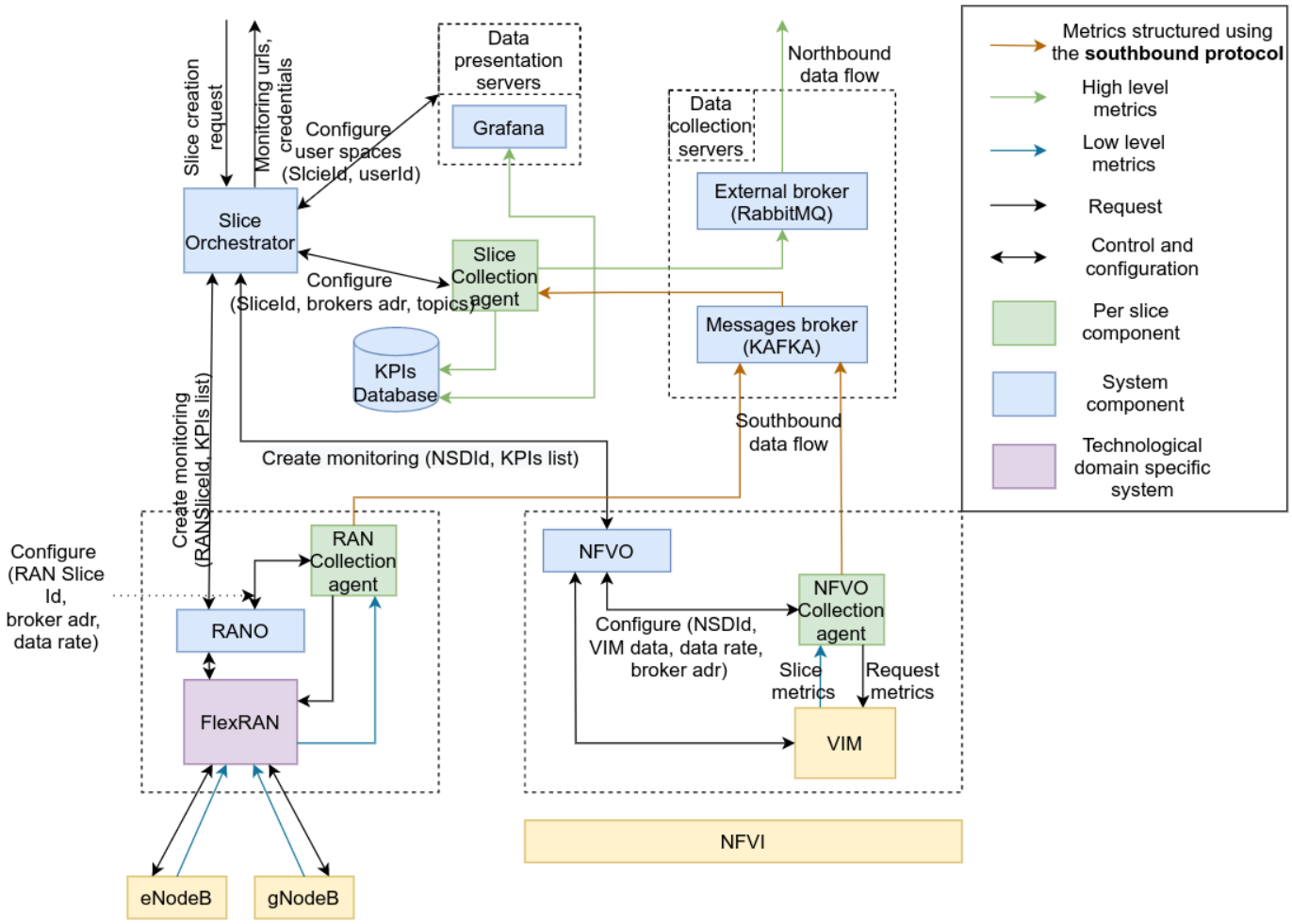

Fig. 2. Architecture of the monitoring system

to first deploy the network slice. Once NFVO and RANO create the sub-slices, an Id for each sub-slice is returned to the SO, namely RANId and NSDId, respectively. SO stores that information on a local DB and links the RANId as well as NSDId to the NSId of the created slice. The second step is the instantiation of the three slice-specific collectors. To do so, SO sends a request to both RANO and NFVO to create the technological domain slice-specific collectors by indicating the sub-slice Id and the list of KPI to collect per domain. Once the RANO and NFVO validate the request, SO instantiates and configures the slice collector agent. The configuration consists of providing information on the Slice ID, the IP addresses of the Brokers, DB where to store the data pushed by the technological domain collectors, and the topics to be used to fetch data at the low-level Broker (i.e., sub-slices ID) as well as topics to publish to at the high-level Broker (i.e., slice ID). SO also needs to inform the presentation server about the creation of a new slice by communicating the Slice ID. As a response, the presentation server communicates a URL and credentials to observe the measured KPI in real-time via a dashboard. SO acknowledges the creation of the network slice and the monitoring system to CSMF by providing (1) URL and credentials for the dashboard; (2) URL and the topic where to fetch raw data of the collected monitoring KPI.

b) RANO: Once receiving the request to create a slice-specific data collector from SO for a sub-slice identified by its RANId, RANO instantiates a slice-dedicated collector agent. The latter is configured with the list of KPI to measure as well as the IP address of the message Broker and the topic to publish to.

To collect KPI from the RAN infrastructure (i.e., eNB/gNB), we rely on the concept of programmable RAN under investigation by the O-RAN initiative [19]. Programmable RAN allows opening RAN through a NBI to be exposed by the eNB/gNB to extract monitoring information or update the configuration on the run-time of eNB/gNB. O-RAN is currently standardizing the different interfaces between the eNB/gNB and a remote RAN controller or a RAN orchestrator. In the proposed framework, we use FlexRAN, a programmable RAN initiative on top of the OpenAirInterface (OAI) eNB/gNB. FlexRAN is composed of an agent sitting at the eNB/gNB. Its role is to extract information from the eNB/gNB, send them to the FlexRAN controller, and enforce at run-time an eNB/gNB configuration policy received from the FlexRAN controller; for instance, to create a new radio slice and assign resources to the network slice. Fig. 3 illustrates the interaction between the RAN data collector and FlexRAN controller as well as

RANO. The RAN data collector periodically requests data on the performance of the sub-slice using the RANId. The collected data includes all information regarding the sub-slices. Therefore, some KPIs can directly be mapped to information provided by FlexRAN, while others need to be obtained by combining different information. The list of RAN KPI supported by our framework is summarized in Table II.

The RAN slice-specific data collector first extracts the data regarding KPI from FlexRAN, or uses a local logic to deduce the KPI if not directly available with FlexRAN. Then, it formats the data by adding meta-data and creates a message using the monitoring protocol described later. Finally, it publishes the message to the low-lever Broker using RANId as a topic.

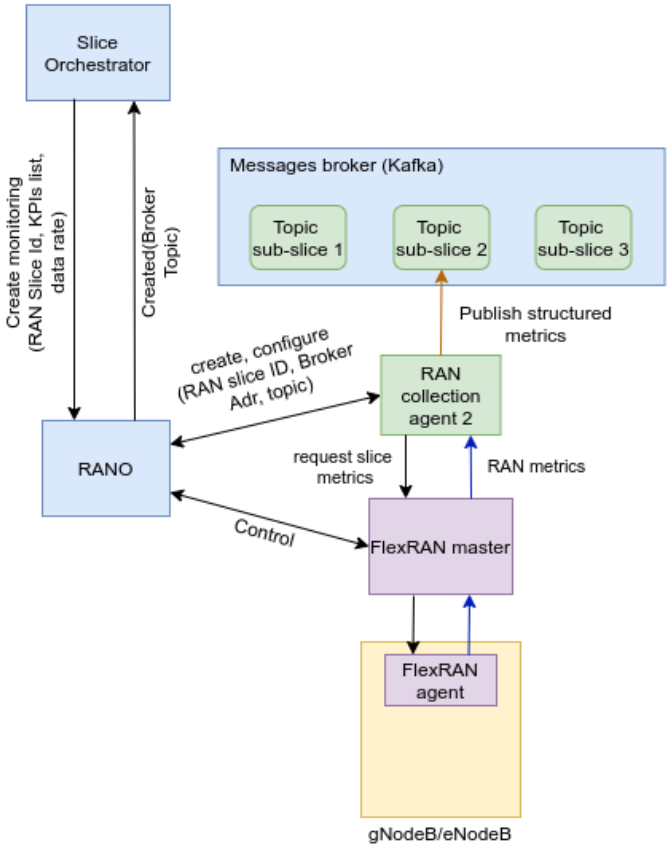

Fig. 3. RAN KPIs collection sub-system

TABLE II
RAN KPIs LIST

KPI	Level	Details
Latency-RAN	RAN	Latency at the RAN (aggregated per slice)
Uplink-data-rate	RAN	Uplink data rate (aggregated per slice)
Downlink-data-rate	RAN	Downlink data rate (aggregated per slice)
Packet-Loss-rate	RAN	Packet loss at the RAN after attempts (RLC layer) (aggregated per slice)
IP-rate	RAN	Packet rate (at PDCP) (aggregated per slice)
Latency-eNB-CN	RAN	Measured RTT between the RAN and CN
Bandwidth	RAN	Bandwidth of cells

c) NFVO: The Cloud/Edge slice-specific data collector shares similar features with the RAN data collector. It is instantiated by NFVO per the request of SO using the NSDId as an identifier. All the collected data on KPI are published on the same message Broker (the low-level) using the NSDId as a topic. It formats data by adding meta-data and generates a message using the same monitoring protocol. In contrast, the Cloud/Edge slice-specific collector relies on other tools to collect KPI on the running sub-slices, particularly for the VNF. Indeed, the KPI to measure for Cloud/Edge domain covers the computing resources and virtual network resources used by the running VNF. Therefore, to collect this data, we rely, in our proposed framework, on the API provided by VIM. However, as we consider in the proposed framework that all VNFs run as Cloud-native Network Function (CNF), i.e., using container-based virtualization, CIS (Container Infrastructure Service) as defined by ETSI [20] replaces the VIM. The CIS is managed by CIS Manager (CISM), which, technologically speaking, corresponds to the Kubernetes[8]. Indeed, Kubernetes provides an NBI that allows collecting several KPIs on the running containers, hence on VNFs. The Cloud/Edge data collector collects the KPI list communicated by the NFVO by consuming the Kubernetes NBI API. It is worth noting that Cloud/Edge sub-slice is composed of a set of VNFs. Thus, NFVO, when instantiating the Cloud/Edge data collector, it communicates the list of VNF instances (Id, names) known by NFVO and linked to the NSD Id. At the CISM level (i.e., Kubernetes), VNFs are identified with their Instance ID. It should be noted that in this work, we relied on Kubernetes NBI to collect KPI, but the platform can use other monitoring collection systems on top of Kubernetes, such as Prometheus. Table III summarizes the list of KPIs that we consider important in the context of network slicing and can be measured by the Cloud/Edge data collector.

TABLE III
NFVO KPIs LIST

KPI	Level	Details
CPU-utilization	NFVO	Aggregated per CNF
Memory-utilization	NFVO	Aggregated per CNF
Number-instances	NFVO	Number of ReplicaSets per CNF
Network-Rx	NFVO	Aggregated per CNF
Network-Tx	NFVO	Aggregated per CNF

B. Data Transfer

1) Monitoring protocol and message format: One of the big challenges that we need to overcome when monitoring the network slice performances (i.e., KPI) is the span of resources over different technological domains. Indeed, each domain has its own system to collect data from the infrastructure. Besides, to the authors best knowledge, there is no existing data collection protocol available in the literature that can address the mentioned concern.Therefore, we propose a novel data collection protocol that defines common monitoring messages for all the technological domains, aiming to abstract lower-level technological domains' infrastructure specificities. Each domain slice-specific data collector will use these messages

[8]https://kubernetes.io/

to encapsulate a monitoring measure and publish it. By doing so, we first simplify the aggregation process to be done by the slice data collector at the SO level. Second, adding a new technological data collector does not impact the other components of the monitoring platform, making it easy to extend the platform in the future. The monitoring communication protocol relies on the Publish/Subscribe concept, where the data collectors produce monitoring data while the slice data collector consumes that data.

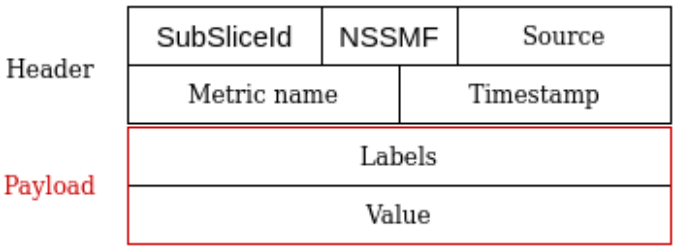

Fig. 4. KPI message structure

All data collected by the technological domain collectors are encapsulated in a common message format shown in Fig. 4. All the fields are detailed in Table IV.

TABLE IV
KPI MESSAGE FIELDS DESCRIPTION

Field	Description	Type
SubSliceId	(required) The Id of the subslice (the slice part) from which the metric was collected.	String
NSSMF	(required) The type of the subslice orchestrator (NFVO, RANO)	String
Timestamp	(required) The time of the metric collection	Long
Metric name	(required) The name of the metric	String
Source	(required) If the Controller Id == NFVO then it should include th VNF name	String
Labels	A set of key-value pairs that help in adding information to the metrics	Key: string value: string, integer, boolean
Value	(required) The value of the metrics	Double, Integer

Each message shall include the sub-slice ID, Controller ID, Timestamp, metric (or KPI) name, and value. The latter corresponds to the measured data. The sub-slice ID indicates the ID of the sub-slice used as a topic for the Publish/Subscribe protocol. The NSSMF field indicates the name of the orchestrator of the technological domain where the data is coming from. The Timestamp field indicates the time when the message is generated. This information is very important as it allows to correlate the KPI coming from different technological domains, which will help to understand the network slice performances' behavior from an end-to-end perspective. Therefore, all the technological domains should be synchronized using the same clock based on GPS and IEEE Precision Time Protocol (PTP). The metric name indicates to which KPI the measured data belongs. The Source field is mandatory only if NSSMF corresponds to NFVO. Indeed, the source field indicates the VNF name concerned with the measured data. The last field, namely label, is not mandatory. It helps to provide multidimensional metrics; for example, if we consider measuring the Channel Quality Indicator (CQI) as calculated by UE in RAN, the label field should include the UE Id (International Mobile Subscriber Identity - IMSI or other identifiers).

2) Data collection servers: The data collection servers are in charge of collecting data from the domain slice-specific data collector deployed for each running network slice. The data collection servers are hard components of the framework as they need to be run in parallel to SO, NFVO, and RANO. The data collection servers use Publish/Subscribe protocol with two levels. The first level (high-level) is allowing the slice owner to consume the monitoring data regarding its run slice in RAW format. The Broker used at this level is based on RabbitMQ, a push-based system. This choice is motivated by the fact that RabbitMQ allows more control over the message routing. It offers more elaborate routing capabilities by providing various exchanges (direct, fan-out, headers, topic). Therefore, using this type of Broker allows the monitoring platform to control what metrics to send to each user and avoid requesting that the consumer manages the messages offset from which it needs to consume. Moreover, the messages are deleted after consumption, which avoids storing redundant data on a metric for a long period of time. Finally, in RabbitMQ, an interesting feature is the possibility of creating a virtual host (vHost) containing the exchanges and their corresponding queues of each user. Those vHosts are used to define the users' permissions and constitute the user space in the Broker. Based on vHosts, we can ensure multi-tenancy and isolation between network slices.

The second level (low-level) is internal to the framework components. It allows collecting the monitoring data generated by the technological domain slice-specific data collector and pushing the data to the concerned slice data collector (at SO level). The Broker at this level is based on Kafka. We argue this choice by the fact that Kafka provides routing by topic, which gives a simple and robust model for internal metrics transfer. Kafka is a pull-based system where the consumers use an offset to access the messages in a topic, and the messages in a topic have a retention period, which allows the consumption of the messages multiple times; this cannot be done using RabbitMQ in which the messages are deleted after consumption. Such a feature allows more control over the consumed messages. SO can change the offset of the messages to consume, hence allowing to collect messages multiple times if needed (in case of database writing problems or collector failure, which results in the deployment of a new collector that can access the metrics that were not handled correctly). Another feature is the notion of partitions within a topic. It allows a group of competitive consumers to get metrics from the same topic in a round-robin way, making the system's scaling easier while keeping a simple model of message routing. Indeed, if the metrics' rate gets higher, new slice collectors can be instantiated for the concerned slice.

C. Data presentation

How data is presented to the user is a very important criterion of a monitoring system. Indeed, the presentation of monitoring data should be adapted to the level of technical knowledge of users. Some users may be satisfied only by visualizing the data through a friendly GUI or dashboard and react to any degradation. Other users may want to have access to the RAW data to store it, and later on, do further analysis using Machine Learning (ML) tools to build models predicting performances or detecting when a parameter is causing an issue. Therefore, the proposed platform covers both ways to present data, i.e., using a GUI and providing RAW data.

Regarding the RAW data, as previously explained, we used the high-level Broker, which is based on RabbitMQ, to expose the collected data, aggregated by the slice data collector. The user can fetch the queue identified by the Slice ID, which stores the monitoring messages. It is then the slice owner's responsibility to write a program or use a RabbitMQ client to connect to the message queue and consume data.

Regarding the graphical presentation, the data presentation server entity is in charge of this task. The data presentation server relies on Grafana[9] to expose via a Dashboard the monitoring messages. Grafana is an open-source, multi-platform, interactive web application for metrics analysis and visualization. It offers fast and flexible visualizations with a multitude of options like tables, graphs, and alerts. A large number of data sources are supported by Grafana, from which we use the InfluxDB[10] data source.

Once the data is available at the lower-level Broker, the slice data collector consumes the message, identifies the subslice, and adds the Slice Id to the metric. The metric then is stored in the KPIs database, and a measurement per metric name is provided and sent to the corresponding topic in the external Broker from which the slice owner can consume it. Grafana is configured when a Network Slice is created by the SO. The SO's monitoring engine creates a dedicated space for the slice owner and its running slices by preparing a folder that contains the dashboards that represent the performances of the user's slices.

It is worth noting that the network operator is granted full access to all the platform's monitoring information. Through Grafana, the network operator can see all the collected monitoring data, aggregated per slice, or aggregated per technological domain. The same information is available as RAW data in the KPI DB that can be used to run ML algorithms for troubleshooting prediction and mitigation, and resource usage optimization.

D. Data privacy and isolation

Multi-tenancy and network slice isolation are an important feature that the devised monitoring framework ensures. It is vital that a slice owner has access to monitored data corresponding to only its running slices. The network slice isolation needs to be mainly enforced when presenting the data to the network slice owners. In the proposed framework, the presentation server and the high-level Broker need to guarantee isolation, as they are interfacing the network slice owners.

In order to ensure data isolation at the high-level Broker, the Slice ID and vHost are used to segregate the monitored data on running network slices. The Slice ID is unique and known only by SO and shared by the latter with the slice owner. The slice owner uses the Slice ID as a topic to fetch the monitoring data. The Slice ID is also communicated to the presentation server along with the User ID (Identifier of the Slice owner). By using a combination of User ID, Slice ID, and vHost, the presentation server can ensure that a slice owner (User ID) can access only data in respect to its running slices (identified via the Slice ID). It is worth recalling that the User ID is communicated by CSMF to SO when creating a network slice. SO stores and associates the User ID with the Slice ID when a network slice is created.

IV. Performance evaluation

A. Testing environment

We have implemented the proposed monitoring framework on top of the 5G facility of EURECOM deployed in the context of the 5GEvE[11] and 5G!Drones[12] projects. EURECOM 5G facility includes all the element introduced in Fig. 1, i.e., SO, RANO and NFVO. Besides, it uses OpenAirInterface (OAI) [13] for the RAN infrastructure and a Openshift/Kubernetes for Cloud/Edge. We have implemented all the components described in Fig. 2. Table V. summarizes the used technologies, which have been adapted and improved.

TABLE V
THE IMPLEMENTED COMPONENTS AND THE USED TECHNOLOGIES

Component	Technology
SO (NSMF)	Python-based
RANO (NSSMF)	Python-based
NFVO (NSSMF)	Python-based
CISM	Kubernetes
Data collection servers	RabbitMQ, Kafka
Data presentation server	Grafana
NBI API	REST

Our experiments were performed on two hosts (Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz 32GB RAM, 6 CPUs without hyper-threading: 1 Thread per core), which form a Kubernetes cluster using Kubeadm v1.19.5. The cluster represents the CIS infrastructure on top of which the cloud sub-slices will be deployed. All the platform components run as containers. It should be noted that the data collector servers (Kafka and RabbitMQ servers), the Data presentation server (Grafana), KPI DB, and the KPIs engine run in the same Kubernetes name-space to measure their resource consumption more precisely.

To get insight into the monitoring platform's performance, we have focused on evaluating two critical aspects. Firstly, we concentrate on the scalability issue when the number of

[9]https://grafana.com/
[10]https://www.influxdata.com/
[11]https://www.5g-eve.eu/
[12]https://5gdrones.eu/
[13]https://openairinterface.org/

running slices is high, where we measured the resource consumption of the whole system when increasing the number of deployed slices. Secondly, we shed-light on the performances of the system, where we measured the latency to present a collected data to the external consumer that a tenant deploys to consume a slice metrics. For all the experiments we used two values of the polling interval (i.e., interval of time to collect data), 1s and 5s. This will allow us to see the impact of polling interval on the measured KPI, knowing that 1sec is very demanding in terms of computing and networking resources. Finally, we collected eight KPI, four on the RAN and four on the Cloud/Edge.

B. System Scalability

The first presented results correspond to the performance of the monitoring system in terms of scalability. To obtain these results, we increase the number of network slices to monitor and measure the consumed computing resources, i.e., CPU and memory. We measured the consumed CPU and memory of the whole monitoring system, but also per component: data presentation, data collection, and the slice-specific collectors.

1) Memory: Fig. 5 represents the RAM consumption of the whole system (including the slice-specific collectors, the data collector serves, and the presentation server) in respect to the number of network slices. As it is expected, the RAM consumption increases (linearly) with the number of network slices to monitor. Besides, when the poling interval is small, the CPU consumption is high. We also remark that when the number of network slices is equal to 50, the RAM consumption exceeds 7 Gb and 8 Gb, for 5s and 1s of polling interval, respectively.

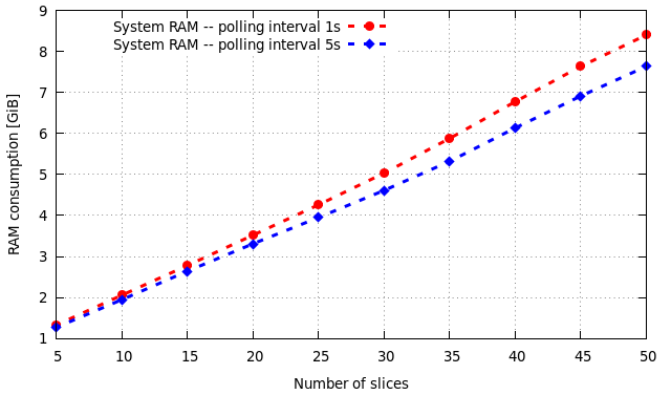

Fig. 5. The RAM consumption of the system as a whole in relation to the number of slices and the polling interval

Now we investigate the RAM consumption per component. Fig. 6 and Fig. 7 illustrate the RAM consumption of the slice-specific data collectors and the data collection servers (Brokers), respectively. Notice that 1 MiB = 2^{20} Bytes and 1GiB = 2^{30} Bytes. It is worth noting that only one curve is shown for both polling intervals as the RAM consumption is merely the same (only 0.1 MiB of difference). This is explained by the fact the poling interval does not impact the consumption of the collectors. Indeed, the memory space used by the collector is not affected by the polling interval, since its role is to request the measurements, structure them, and send them to the Brokers of the data collection servers, which do not require high computing resources. We recall that three collectors are instantiated per slice; one at SO, and one for each technological domain. We remark that the slice-specific collectors are consuming merely 80% of the RAM of the whole system. Obviously, the RAM consumption increases linearly with the number of monitored network slices; the RAM consumption reaches 6 GiB for a high number of monitored network slices. On the other hand, we remark that the brokers consume less RAM; less than 2 GiB and 2.4 GiB, when the number of slices is equal to 50 and for a polling interval of 5s and 1s, respectively.

For the Data presentation server, the RAM consumption is practically constant and is around 20 MiB.

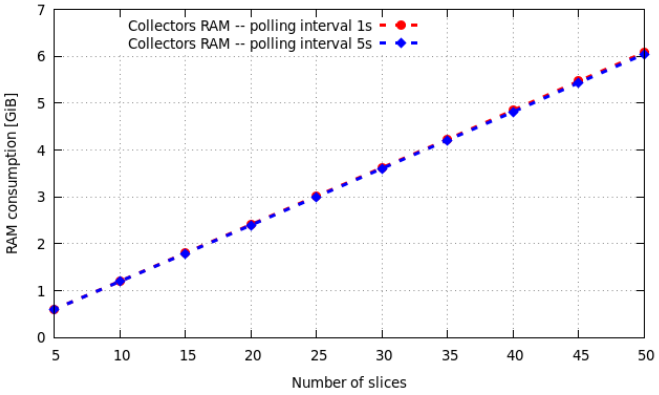

Fig. 6. The RAM consumption of the slice-specific data collectors in relation to the number of slices and the polling interval

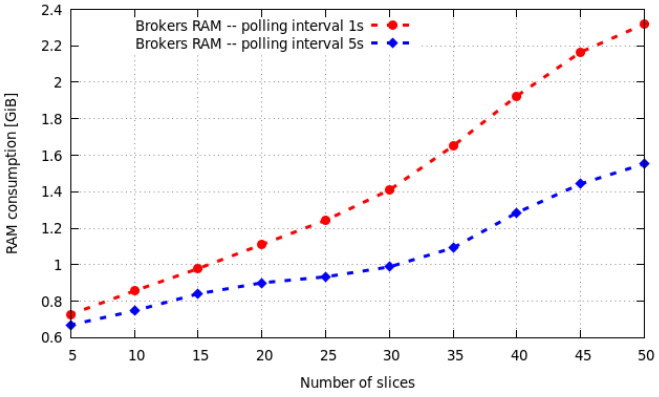

Fig. 7. The RAM consumption of the data collection servers (data brokers) in relation to the number of slices and the polling interval

2) CPU: To calculate the CPU consumption of the monitoring system elements, we rely on cAdvisor[14], which is a daemon that collects, aggregates, processes, and exports information such as resource usage and performance characteristics about running containers. CAdvisor is integrated into Kubernetes and provides a "container_cpu_usage_seconds_total"

[14]https://github.com/google/cadvisor

metric that shows the cumulative CPU time consumed by a given container. This metric is scrapped periodically using Prometheus. Then, we apply a rate on the time series, i.e., we use a function that calculates the per-second average rate of increase of the time series in the given time interval, representing the CPU consumption of a container. Using this way of computing CPU explains the decimal values of CPU shown in the figures.

Fig. 8 shows the entire monitoring system CPU consumption with respect to the number of monitored network slices for the two polling intervals. Clearly, we observe a similar behavior as for the RAM, where the CPU consumption increases with the increase of the number of monitored slices. Besides, the CPU consumption is higher when the polling interval is small; it reaches 4.5 CPU and 3 CPU for 1s and 5s polling interval respectively while the number of monitored slices is 50. This means that in a saturated situation, the whole system uses merely 5 CPU and 9 GiB of RAM, which is an acceptable value in modern hardware.

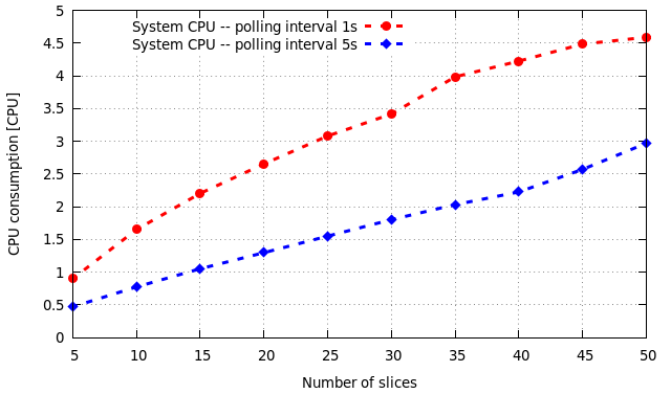

Fig. 8. The CPU consumption of the system as a whole in relation to the number of slices and the polling interval

As for the RAM, we will investigate which components are consuming more CPU. Fig. 9 and Fig. 10 show CPU consumption of the slice-specific collectors and the data collection servers, respectively. As for the RAM, we remark that the slice-specific collectors consume more CPU, reaching 2.5 CPU and 3.7 CPU when the number of monitored network slices is 50 and for a polling interval of 5s and 1s, respectively. We also observe that reducing the polling interval has a strong impact on CPU consumption. When the number of monitored slices is high, the difference could reach merely a double. Meanwhile, the difference is less obvious for the Brokers, where CPU consumption is not highly impacted by the polling interval. We remark the same behaviour when increasing the number of monitored slices; i.e., a small impact on CPU consumption. For instance, for a polling interval of 1s, the difference of consumed CPU between 10 monitored slices and 50 is around 0.4 CPU. This indicates that the Brokers scale well with the number of network slices.

Regarding the data presentation server the CPU usage is very low and is around 0.004 CPU.

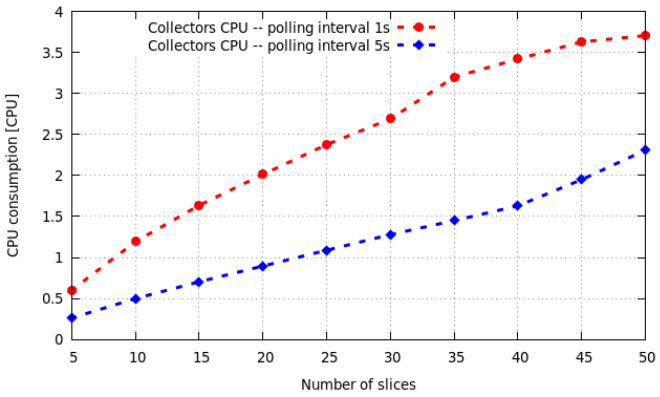

Fig. 9. The CPU consumption of the slice-specific data collectors in relation to the number of slices and the polling interval

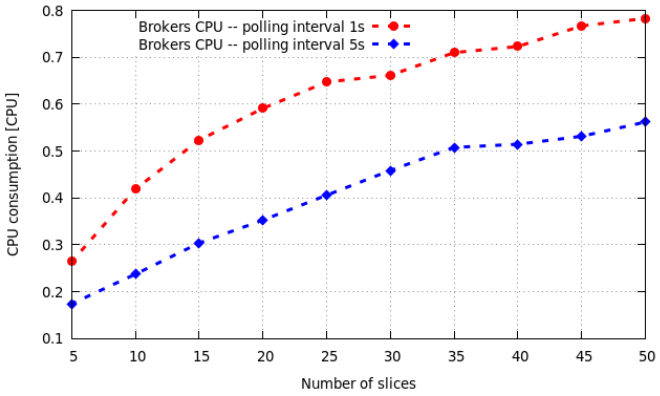

Fig. 10. The RAM consumption of the data collection servers (data brokers) in relation to the number of slices and the polling interval

C. End to end messages latency

Now we turn our attention to the performance of the monitoring system in terms of latency to deliver the monitored data to the slice owner. To this aim, we measured the time taken by the system to collect data and present them to the slice owner. We measure this latency for the two data collection polling intervals while increasing the number of monitored network slices. We clearly observe that the latency increases linearly in the case of a polling interval of 5s but exponentially for 1s. It reaches 160 ms and around 40 ms for a polling interval of 1s and 5s, respectively. This clearly proves the strong impact of the polling interval on the latency. However, it remains acceptable, less than 1s in each case, and even when the number of monitored network slices is very high.

V. Conclusion

In this paper, we introduced a novel monitoring framework for network slicing in 5G. The proposed framework solves many issues that arise when monitoring the performances of network slices. First, it is a scalable framework, as slice data collectors are instantiated within a network slice and deleted when the network slice ends. Second, it allows monitoring resources of different technological domains and abstracts each domain's specificity by devising a novel data collection

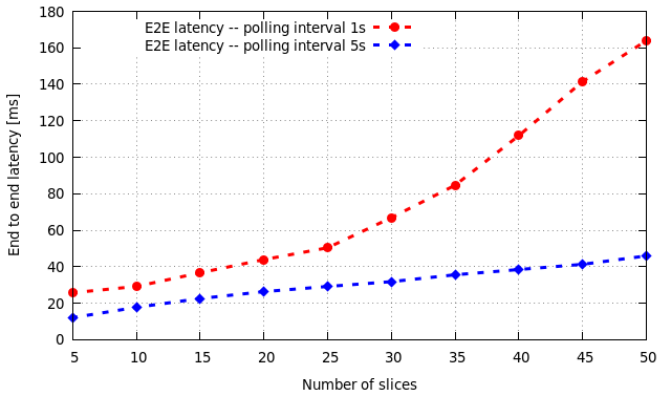

Fig. 11. The end-to-end messages latency in relation to the number of slices and the polling interval

protocol. Third, aggregate the measured data at the slice level and provided it to the slice owner as raw data or via a graphical interface. The proposed framework has been implemented in a 5G facility and extensively evaluated. Obtained results indicate that even if a high number of network slices are deployed and monitored, the CPU and memory consumption remain sustainable by current hardware. In addition, the latency to present the data to the slice owner remains under 1s when a high number of network slices are deployed.

VI. Acknowledgment

This work was partially supported by the European Union's Horizon 2020 Research and Innovation Program under the 5G!Drones project (Grant No. 857031).

References

[1] "Sma public policy position," *(2020, March) 5G Spectrum*.

[2] A. Ksentini and P. A. Frangoudis, "Toward slicing-enabled multi-access edge computing in 5g," *IEEE Network*, vol. 34, no. 2, pp. 99–105, 2020.

[3] A. Ksentini, P. A. Frangoudis, P. Amogh, and N. Nikaein, "Providing low latency guarantees for slicing-ready 5g systems via two-level mac scheduling," *IEEE Network*, vol. 32, no. 6, pp. 116–123, 2018.

[4] B. Brik and A. Ksentini, "On predicting service-oriented network slices performances in 5g: A federated learning approach," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*. IEEE, 2020, pp. 164–171.

[5] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of cloud monitoring tools: Taxonomy, capabilities and objectives," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2918–2933, 2014.

[6] R. Perez, J. Garcia-Reinoso, A. Zabala, P. Serrano, and A. Banchs, "A monitoring framework for multi-site 5g platforms," in *2020 European Conference on Networks and Communications (EuCNC)*. IEEE, 2020, pp. 52–56.

[7] X. Vasilakos, B. Köksal, D. H. Izaldi, N. Nikaein, R. Schmidt, N. Ferdosian, R. F. Sari, and R.-G. 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, 2020, pp. 1–7.

[8] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "Flexran: A flexible and programmable platform for software-defined radio access networks," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 427–441.

[9] A. Beltrami, P. D. Maciel, F. Tusa, C. Cesila, C. Rothenberg, R. Pasquini, and F. 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.

[10] F. Tusa, S. Clayman, and A. Galis, "Dynamic monitoring of data center slices," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 286–290.

[11] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with lattice," in *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*. IEEE, 2010, pp. 239–246.

[12] M. B. de Carvalho, R. P. Esteves, G. da Cunha Rodrigues, L. Z. Granville, and L. M. R. Tarouco, "A cloud monitoring framework for self-configured monitoring slices based on multiple tools," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*. IEEE, 2013, pp. 180–184.

[13] S. Kukliński and L. Tomaszewski, "Dasmo: A scalable approach to network slices management and orchestration," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–6.

[14] I. Afolabi, T. Taleb, P. A. Frangoudis, M. Bagaa, and A. Ksentini, "Network slicing-based customization of 5g mobile services," *IEEE Network*, vol. 33, no. 5, pp. 134–141, 2019.

[15] *3rd Generation Partnership Project (3GPP), 2018b. "Study on management and orchestration of network slicing for next generation network". 3GPP TS 28.801 version 15.1.0 Release 15.*

[16] "Generic network slice template", version 3.0," *NG.116,May 2020*.

[17] C.-Y. Chang, N. Nikaein, O. Arouk, K. Katsalis, A. Ksentini, T. Turletti, and K. Samdanis, "Slice orchestration for multi-service disaggregated ultra-dense rans," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 70–77, 2018.

[18] *Multi-access Edge Computing (MEC); MEC Management; "Part 1: Application lifecycle, rules and requirements management", ETSI GS MEC 010-1 V1.1.1 (2017-10).*

[19] *ORAN Alliance, 2020b. "Operator defined next generation ran architecture and interfaces". URL:https://www.o-ran.org/.*

[20] *Network Functions Virtualisation (NFV) Release 3; Virtualised Network Function; "Specification of the Classification of Cloud Native VNF implementations", ETSI GS NFV-EVE 011 V3.1.1, Oct 2018.*

Mohamed Mekki received his engineering degree in Computer Systems from the Higher School of Computer Science in Algiers, Algeria in 2020. He is currently a 1st-year doctoral student at EURECOM, Sophia Antipolis. His doctoral thesis is on Cloud Edge Continuum (CEC) to support emerging network services that require low latency and high bandwidth usage.

Sagar Arora (sagar.arora@eurecom.fr) received his Masters in Mobile Communication from Eurecom, Sophia Antipolis in 2019. He is currently a 2nd-year doctoral student at EURECOM, Sophia Antipolis. His doctoral thesis is on designing a network slice orchestration framework for cloud-native container based 5G network functions and MEC applications.

Adlen Ksentini is a COMSOC distinguished lecturer. He obtained his Ph.D. degree in computer science from the University of Cergy-Pontoise in 2005, with a dissertation on QoS provisioning in IEEE 802.11-based networks. From 2006 to 2016, he worked at the University of Rennes 1 as an assistant professor. During this period, he was a member of the Dionysos Team with INRIA, Rennes. Since March 2016, he has been working as a professor in the Communication Systems Department of EURECOM. He has been involved in several national and European projects on QoS and QoE support in future wireless, network virtualization, cloud networking, mobile networks, and more recently on Network Slicing and 5G in the context of H2020 projects 5G!Pagoda, 5GTransformer, 5G!Drones and MonB5G. He has co-authored over 120 technical journal and international conference papers. He received the best paper award from IEEE IWCMC 2016, IEEE ICC 2012, and ACM MSWiM 2005. He has been awarded the 2017 IEEE Comsoc Fred W. Ellersick (best IEEE communications Magazine's paper). Adlen Ksentini has given several tutorials in IEEE international conferences, IEEE Globecom 2015, IEEEE CCNC 2017, IEEE ICC 2017, IEEE/IFIP IM 2017. Adlen Ksentini has been acting as TPC Symposium Chair for IEEE ICC 2016/2017, IEEE GLOBECOM 2017, IEEE Cloudnet 2017, and IEEE 5G Forum 2018. He is in the editorial board of IEEE Network and IEEE Networking Letters. He acted as Guest Editor for IEEE Journal of Selected Area on Communication (JSAC) Series on Network Softwerization, IEEE Wireless Communications, IEEE Communications Magazine, IEEE Transactions on Network Science and Engineering (TNSE), and two issues of ComSoc MMTC Letters. He has been on the Technical Program Committees of major IEEE ComSoc, ICC/GLOBECOM, ICME, WCNC, and PIMRC conferences. He acted as the Director of IEEE ComSoc EMEA region and member of the IEEE Comsoc Board of Governor (2019-2020). He was the chair of the IEEE ComSoc Technical Committee on Software (TCS) (2019-2020).