Received: 20 December 2022 | Revised: 24 May 2023 | Accepted: 26 June 2023

DOI: 10.1049/ntw2.12095

IET Networks

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Deploying cloud-native experimental platforms for zero-touch management 5G and beyond networks

Sergio Barrachina-Muñoz[1] | Rasoul Nikbakht[1] | Jorge Baranda[1] | Miquel Payaró[1] | Josep Mangues-Bafalluy[1] | Panagiotis Kokkinos[2] | Polyzois Soumplis[2] | Aristotelis Kretsis[2] | Emmanouel Varvarigos[2]

[1]Services as Networks (SaS) at Centre Tecnològic Telecomunicacions Catalunya (CTTC/CERCA), Barcelona, Spain

[2]Department of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece

Correspondence

Sergio Barrachina-Muñoz.
Email: sbarrachina@cttc.cat

Funding information

MARSAL, Grant/Award Number: 101017171; MINECO, Grant/Award Numbers: TSI-063000-2021-56, TSI-063000-2021-57

Abstract

An experimental framework for managing 5G and beyond networks through cloud-native deployments and end-to-end monitoring is presented. The framework uses containerised network functions in a Kubernetes cluster across a multi-domain network spanning cloud and edge hosts. End-to-end monitoring is demonstrated through Grafana dashboards that showcase both infrastructure resources and radio metrics in two scenarios involving UPF re-selection and user mobility. As a third scenario, the authors demonstrate how a decision engine interacts with the experimental platform to perform zero-touch containerised application relocation, highlighting the potential for enabling dynamic and intelligent management of 5G networks and beyond.

KEYWORDS

5G mobile communication, computer networks, decision making, mobile computing, radio access networks, virtualisation

1 | INTRODUCTION

5G and beyond (B5G) networks are set to address the demands of a fully connected and mobile society, enabling a wide variety of services and applications over the same infrastructure. Further, 5G adopts edge computing as a key paradigm, evolving from centralised architectures towards multiple points-of-presence (PoPs) of edge nodes. In turn, this enables multi-access edge computing (MEC) applications for low-latency and high bandwidth like virtual and augmented reality (VR/AR). In this context, projects like MARSAL [1, 2] propose a new paradigm of elastic virtual infrastructures that integrate transparently a variety of novel radio access, networking, management, and security technologies to deliver end-to-end transfer, processing, and storage services in an efficient and secure way.

To materialise this vision for 5G and B5G networks, three elements are key: (*i*) cloud-native deployments within the network function (NF) virtualisation (NFV) paradigm, (*ii*) end-to-end monitoring, and (*iii*) intelligence to efficiently capitalize on the data gathered via monitoring to perform the best zero-touch network management decision. As for the former, cloud-native infrastructures make it possible to share infrastructure resources, enabling their dynamic allocation to meet the service level agreements (SLAs) of existing and future demanding use cases. Cloud-native technologies also reduce time to market, respond sooner to customer demands, and facilitate the lifecycle management and automation of the network. Therefore, they are widely regarded as the future of vertical application development with enhanced flexibility, scalability, and reduced cost.

Monitoring is the second key aspect this paper deals with. Indeed, a paramount factor for 5G is end-to-end real-time monitoring, gathering infrastructure metrics (i.e., compute, storage, and network) as well as domain-specific metrics of components such as gNBs or MEC services. Gathering these

This work is an extension of our CSNDSP'22 conference paper.

metrics supports the lifecycle management of services running over the 5G network and favours intelligent reconfiguration and alerting to involved tenants and stakeholders, spanning from infrastructure owners, operators, slice owners, or service/application developers. Certainly, multi-tenant networks must support network slices, which require monitoring key performance indicators (KPIs) commonly belonging to different technological domains and managed by different entities. For instance, a network operator shall focus on the KPIs of the components running the network slice (e.g., RAN, cloud/edge, routers), while the slice owner may consider high-level KPIs (e.g., end-to-end delay) for SLA validation [3]. In both cases, monitoring turns into an imperative aspect.

The third element of chief importance is intelligence via what has been coined as a decision engine (DE). By utilising a DE that has access to monitored metrics (e.g., CPU consumption) plus additional constraints that can be added as input (e.g., cost of hosting an application), the most appropriate node can be selected for deploying a given application. This can lead to improved performance, reduced latency, and cost savings, making it an attractive option for managing large-scale cloud-native network deployments. In addition, a DE can help to ensure that the deployment of an application is aligned with the overall goals and constraints of the infrastructure manager, such as minimising costs or maximising performance.

Even though there are valuable works in the literature on cellular networks devoted to cloud-native deployments, monitoring, and the intelligence behind it, only a few of them (partially) treat the three aspects together. In contrast, this paper deals with the three aspects by presenting a cloud-native experimental platform for edge-enabled 5G and beyond networks endowed with an end-to-end monitoring system and a DE to make the most efficient use of the gathered data. The contributions can be summarised as follows:

- Cloud-native 5G core deployment with Open5GS in a multi-PoP Kubernetes cluster, where each NF runs as a separate containerised NF (CNF).
- End-to-end containerised monitoring gathering both infrastructure and radio/RAN metrics via CNFs.
- Integration of a commercial gNB (Amarisoft Callbox) into the 5G testbed and development of a custom sampling function for pulling gNB metrics (e.g., downlink/uplink bit rate).
- Showcasing of the monitoring system through the visualisation of different metrics in Grafana dashboards. Two toy scenarios are considered; one for UPF re-selection and the other for UE mobility.
- Intelligence layer (DE) on top of the experimental infrastructure that enables the best possible placement for applications and NFs based on the data gathered by the monitoring system.

The rest of the article is structured as follows. Section 2 discusses the related work, while section 3 describes the main components of the experimental platform and the edge-enabled testbed. Next, section 4 depicts the end-to-end monitoring system and section 5 depicts the DE. Then, section 6 showcases and validates the whole framework. Finally, conclusions are collected in section 7.

2 | RELATED WORK

There are several works in the literature separately dealing with cloud-native 5G deployments, end-to-end monitoring systems for 5G networks, or intelligent algorithms for the placement of applications and network functions. However, none of them fully address the three aspects together as we discuss what's next.

As for cloud-native deployments, a method to orchestrate and manage a container-based C-RAN using Kubernetes and OpenAirInterface (OAI) is presented in ref. [4]. Authors in ref. [5] introduce an open-source infrastructure for 5G RAN development, where DevOps simplifies the deployment of end-to-end applications to the edge. Also, Kube5G is proposed in ref. [6] for building and packaging a cloud-native telco NF through nested layers, and a 5G cloud-native environment based on Kubernetes and Openshift Operator is introduced in ref. [7]. Recently, an integration of KubeFed for deploying workloads in multiple clusters and Network Service Mesh for providing connectivity across cluster boundaries has been proposed in ref. [8]. The aforementioned works make valuable efforts towards cloud-native deployments. However, none of them realize a full 5G core, but different variations of 4G's evolved packet core (EPC). Further, they do not focus on monitoring.

As for papers dealing with monitoring, authors in ref. [9] present SONATA, a multi-PoP monitoring framework of NFV services involving both containers and virtual machines. However, monitoring is discussed from an architectural perspective, and no actual 5G core is deployed. Instead, authors in refs. [10, 11] present an approach to deliver monitoring and telemetry mechanisms as a service using Prometheus and Netdata over an Open5GS network, but no containerisation is provided. Similarly, a monitoring framework introducing metrics collectors deployed per network slice using Prometheus is devised in ref. [3], but no containerised deployment of the 5G core is provided either. In ref. [12], authors investigate the effect of inter-NF dependencies in terms of resource consumption in a Free5GC network deployed in Kubernetes. However, only limited monitoring (e.g., RAN parameters are not considered) is undertaken through custom Python scripts. Finally, ref. [13] introduces the 5GROWTH service platform with an AI-driven automated 5G end-to-end slicing. However, no 5G core is deployed.

Finally, regarding intelligent algorithms for optimal application and function placement, these are often posed as resource allocation problems. The resource allocation problem in virtualized environments is a multi-dimensional research area that has attracted the interest of the research community. The modelling of the problem among the different works varies according to the considered topology and the adopted

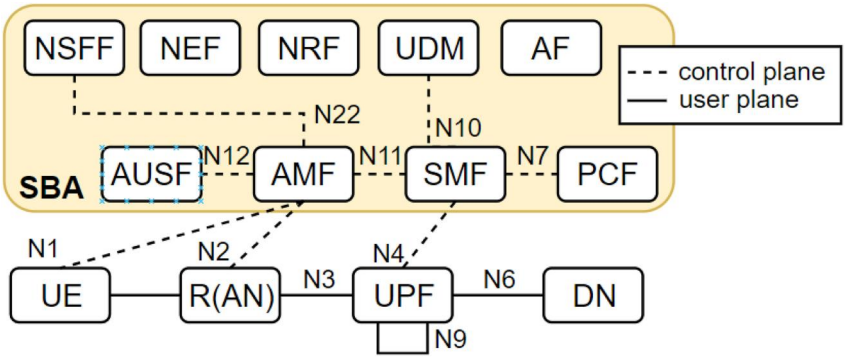


FIGURE 1 5G System Architecture in reference point representation (based on [19]). Some NFs and interfaces are not included for the sake of representation.

technologies while the proposed solutions employ techniques from the wider realm of mathematics and computer science. For example, ref. [14] examines the placement of virtual machines (VMs) on top of physical systems in a cloud data centre to per [15] focus on the optimal placement of VNFs (Virtualized Network Functions) in SDN-NFV enabled Multi-Access Edge Computing (MEC) nodes with the aim of minimising the deployment and resource usage cost. In ref. [16] the microservices' placement in a multi-layered fog/edge environment is considered, targeting to place them as close as possible to the data sources.

Unlike the previous works, the framework proposed in this paper jointly provides full cloud-native deployment of both 5G core and end-to-end monitoring (including RAN) through CNFs orchestrated via Kubernetes. Besides, the framework is visually demonstrated through three scenarios reflecting a series of events in the context of (i) UPF re-selection, (ii) user mobility, and (iii) application placement (which capitalises the intelligence of the decision engine).

3 | CLOUD-NATIVE 5G EXPERIMENTAL PLATFORM

The 5G architecture consists of two parts that have remarkably changed from previous generations: the new radio access network (NG-RAN) supporting the new radio (NR) and the 5G Core Network (5GC). In this section, we describe the main components of our experimental platform [17] and propose an edge-enabled testbed to demonstrate the end-to-end monitoring system.

3.1 | Open-source 5G core

The movement towards softwarisation of telecommunication networks has deeply influenced the creation of 5GC [18]. Rather than relying on monolithic elements, 5G adopts a service-based architecture (SBA) composed of NFs that modularise the tasks of the core. As shown in Figure 1, these NFs interact through Service-Based Interfaces (SBIs), which employ Representational State Transfer (REST) interfaces. A key feature of such SBA modularisation is network slicing, which benefits from softwarisation and cloudification. In essence, slices represent logical instances of the network that can be tailored to optimise services and thus cope with different service level agreements (SLAs) according to the use case. Further, within the SBA, we may find the NWDAF (Network Data Analytics Function) and the MDAF (Management Data Analytics Function) for generating insights from NFs data and taking actions to enhance performance, including slice selection and control [20].

This new architecture allows 5G stakeholders much more flexibility and openness, paving the way for an environment where open-source perfectly suits. In this regard, some alternatives of open-source 5GC are available, such as OpenAirInterface [21] CN, Free5GC [22], and Open5GS [23]. In our experimental framework, we use Open5GS v2.4.0, since it includes most of the 5GC NFs defined in 3GPP and also allows deploying more than one UPF instance, thus supporting edge-enabled networks. Nevertheless, our experimental framework is designed to also work with other open-source 5G cores under acceptable adjustments.[1]

3.2 | Cloud-native deployment of the 5G core

5G is expected to support use cases that go beyond raw throughput performance, where the focus is to be put on service flexibility and agility. In this regard, the procedure to deploy 5G NFs has a critical impact. In essence, these functions can be instantiated as physical NFs (PNFs), virtual NFs (VNFs), or containerised NFs (CNFs). Naturally, VNFs gained momentum against siloed PNFs since the conceptualisation of the modularised 5G SBA because of the virtualisation benefits in terms of efficiency, scalability, or cost. Recently, it is the turn of Containerised Network Function (CNFs) to gain momentum among operators [24] against conventional Virtualized Network Function (VNFs) due to their higher degree of scalability, efficiency for operation and management, energy-saving, and suitability for resource-constrained edge applications.

[1]Preliminary deployments of our containerized framework with OpenAirInterface CN and Free5GC have been also already successfully validated.

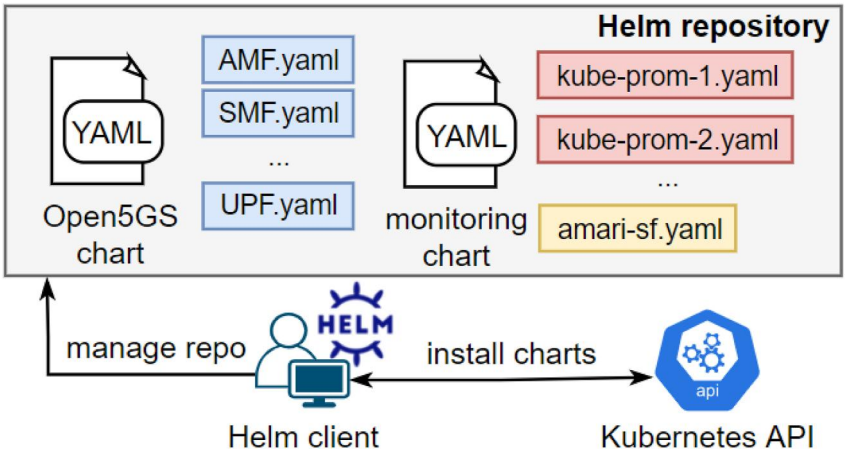


FIGURE 2 Helm and Kubernetes flowchart. The Open5GS chart is composed of the NF templates in blue (e.g., AMF), whereas the monitoring chart includes templates for kube-prometheus in red and the Amarisoft sampling function in yellow.

According to Docker [25], a container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. This makes a container image a lightweight, standalone, executable package of software that includes everything needed to run an application. Container deployments, which may span multiple hosts, are managed with an orchestrator responsible for automating container creation, deletion, and modification without service disruption. Notice that such tasks on containers match the NFV lifecycle management. In this work, we adopt Kubernetes as a container orchestrator, since it is the de facto solution in multiple industries for high-demand services with complex configurations.

As explained in the following subsections, we deploy both the 5GC NFs and monitoring system within the same Kubernetes cluster using Helm charts,[2] a collection of files that describe a related set of Kubernetes resources (see Figure 2).[3] This way, the whole framework is deployed in just two commands, one for installing the monitoring system and the other for installing Open5GS. Once the deployments are instantiated, connectivity among containers and towards external services must be provided. In particular, our Kubernetes cluster relies on Calico [27], a well-known container network interfaces (CNIs) plugin to implement such networking capabilities.

3.3 | Experimental RAN integration

The radio access network (RAN) is another critical component of 5G networks, since it provides individual users with wireless connectivity to the core and external data networks. There are different alternatives when it comes to experimenting with 5G RAN, which can be classified in simulated/emulated (e.g, UERANSIM [28]) and physical/real (e.g., Amarisoft). In this work, in order to provide actual over-the-air-transmissions, we rely on Amarisoft's AMARI Callbox Ultimate [29] acting as a gNB with high-performing NR capabilities. Nevertheless, other RAN alternatives like UERANSIM have been also successfully integrated within the testbed without requiring any configuration changes in the Open5GS Helm chart.

Notice that the RAN in our framework is essentially a physical NF (PNF), whereas the 5GC and monitoring system are built on CNFs. As for how to integrate the AMARI gNB with the Open5GS core, we shall indicate the AMF's IP address in the gNB configuration file to establish the NG Application Protocol (NGAP) connection. In our case, since the AMF runs as a CNF, a custom Kubernetes service exposes the AMF functionality to let the gNB point to the master node IP rather than to the AMF's pod IP. This is a common practice, since pod IPs may change after deletion, while services remain fixed.

3.4 | An edge-enabled 5G testbed

Testbeds are essential in telco research, as new architectures, techniques, and features can be conveniently assessed and validated in the lab before going into field trial campaigns. In this work, we implement and integrate the testbed shown in Figure 3 to showcase our cloud-native end-to-end 5G experimental platform. In particular, the 5G network is composed (from left to right) of the following elements: two UEs emulated with Amarisoft AMARI UE Simbox Series [30], with UE1 always targeting best-effort services while UE2 may target both best-effort and time-critical MEC applications (e.g., AR/VR); an Amarisoft Callbox acting as a stand-alone 5G gNB; an edge node running the MEC UPF, an iperf server, and an ETSI MEC API [31]; a core node running the Open5GS CNFs and another iperf server; a monitoring node hosting the monitoring containers; and a master node managing the Kubernetes cluster. Since the focus of our current testbed implementation is on the monitoring of the RAN and core domain components, the access and transport networks have been simplified

[2] All the source code of our experimental platform [17, 26] is open.

[3] Notice that all the NFs in Open5GS (AMF, AUSF, BSF, NRF, NSSF, PCF, SMF, UDM, UDR, and UPF) can be compiled and deployed separately, making it a suitable candidate for evaluating the performance of distributed and cloud-native deployments of the 5GC.

to Ethernet links in a local area network. As shown in the Kubernetes representation in Figure 4, the cluster is composed of the aforementioned nodes (master, core, edge, and monitoring). Therefore, the master is responsible for deploying, controlling, deleting, and updating the containers of each of the nodes.

Remarkably, all the 5GC NFs are deployed in one click (through Helm) in the core and edge host. In particular, the core runs all the Open5GS NFs available in v2.4.0, including both critical (e.g., AMF or SMF) and secondary NFs (BSF). Besides, for the sake of enabling MEC platforms, the presented cloud-native deployment provides two UPFs: one located at the core and the other, at the edge node. This way, each UPF serves as a PDU session anchor and provides a connection point to different access networks, one being the conventional Internet, and the other any data network that can benefit from MEC processing capabilities, like AR/VR application components.

Finally, the monitoring node is in charge of running all the containers related to the end-to-end monitoring of the components. To that aim, and as explained in §IV, kube-prometheus [32] is deployed for monitoring Kubernetes elements, while a custom sampling function is developed to pull metrics from the Amarisoft Callbox API. So, these monitoring CNFs can be viewed as 5G's application functions (AFs) within 5G's SBA.

4 | END-TO-END MONITORING: FROM CORE TO RAN

We depict below the monitoring system designed for our 5G experimental framework. We shall emphasise two main characteristics that make it a valuable asset: it is cloud-native, meaning that specific CNFs are deployed for monitoring purposes, and it is end-to-end, meaning that deployed CNFs

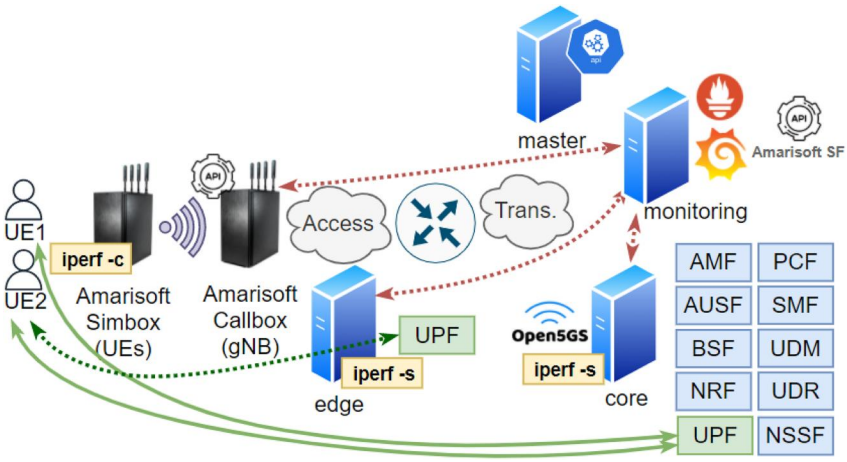


FIGURE 3 Edge-enabled 5G testbed. Data planes are shown in green (dashed arrow for the edge data plane). Monitoring connections are represented with red dashed arrows.

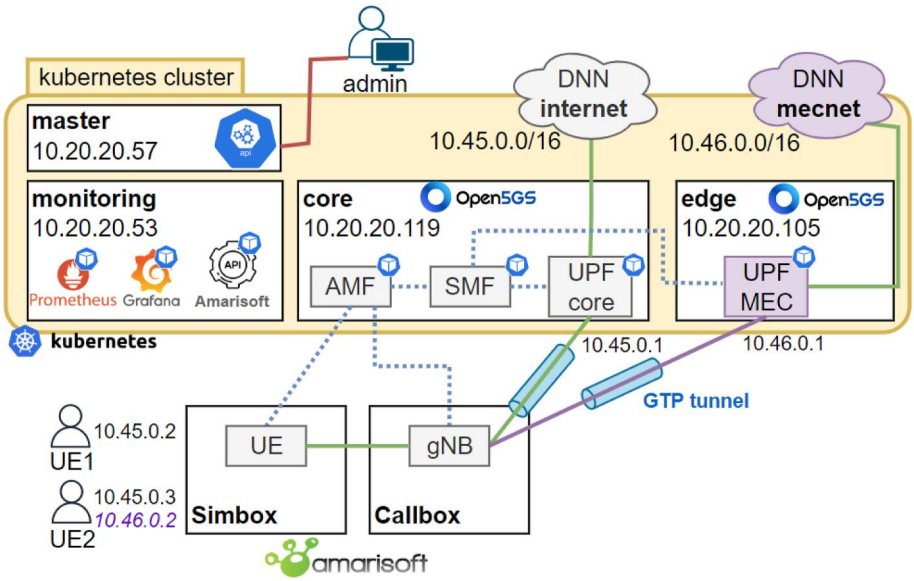


FIGURE 4 Kubernetes deployment. Notice that UE2 gets an IP address depending on its assigned UPF, that is, 10.45.0.3 and 10.46.0.2 for the core UPF and MEC UPF, respectively.

pull metrics both from the core and the RAN domains. The data is gathered into a centralised database where metrics are then plotted in dashboards.

4.1 | Monitoring with prometheus

Monitoring is the practice of examining behavioural data from infrastructure, network events, and user interactions. Thus, in order to let network administrators gather metrics of interest and manage unexpected events, a proper monitoring system must be able to collect data from multiple sources.

In this work, we primarily rely on Prometheus [33], since it is an open-source monitoring and alerting toolkit that can be easily integrated with Kubernetes to support automatic deployment, letting agents be automatically discovered via service discovery. The Prometheus server also opens interfaces to third-party applications, like web UI or Grafana. We refer the reader to ref. [34] for a comparison on monitoring tools focused on 5G networks. In particular, we use Prometheus in two different approaches to tackle infrastructure and RAN monitoring. For the former, we use kube-prometheus [32] and rely on a custom sampling function for the latter.

4.1.1 | Kube-prometheus for infrastructure

To monitor the usage of infrastructure resources, such as compute, storage, and network, we rely on kube-prometeus stack, an easy to operate end-to-end Kubernetes cluster monitoring stack that uses Prometheus Operator. The stack is pre-configured to collect metrics from all Kubernetes components – meaning resources are measured at different levels, such as pod, workspace, Kubernetes node, or host – and it also delivers a default set of dashboards and alerting rules. Therefore, we can deploy a full off-the-shelf cluster monitoring tool with a single Helm command. Among the metrics gathered with kube-prometheus, we find CPU, memory, transmit/receive networking, etc. Some of them are presented at node level in 6.1.

4.1.2 | Custom sampling function for RAN

For our monitoring purposes, the Amarisoft Callbox can be accessed through a remote API using the WebSocket protocol, which establishes a persistent connection between the client (Amarisoft sampling CNF in monitoring node) and the server (Amarisoft Callbox itself). This API exposes different metrics at gNB/radio level, including (per user and cell id) uplink and downlink bitrate, modulation coding scheme (MCS), channel quality indicator (CQI), or signal-to-noise-ratio (SNR). The custom sampling function we developed is a containerised Python script that opens a WebSocket against the Callbox API and exposes some metrics of interest to the Prometheus scraper. We showcase some of these RAN metrics in 6.2.

4.2 | Visualising metrics with grafana

Once the data is being gathered, it is usually convenient to visualise it to quickly grasp the behaviour of the network at different domains. For such a task, we use Grafana, a multi-platform open-source analytics and interactive visualisation web application that is also included in the kube-prometheus stack. We modified the corresponding chart with the inclusion of two Grafana dashboard descriptors in JSON format for the experiments in 6.1 and 6.2, respectively.

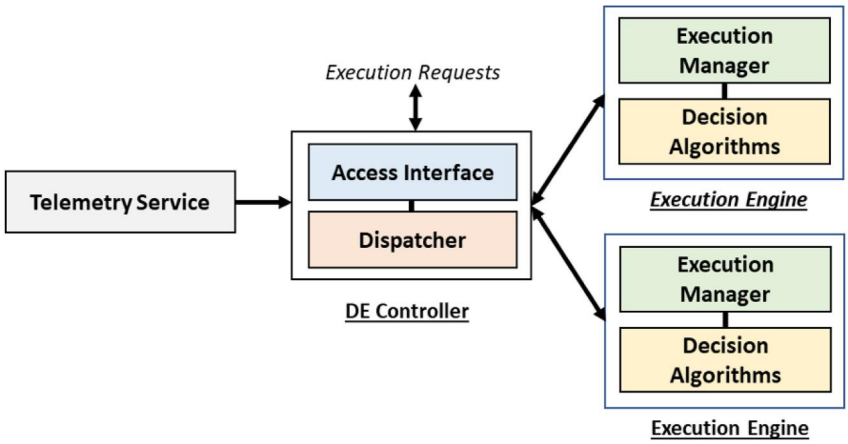
5 | DECISION ENGINE FOR ZERO-TOUCH MANAGEMENT

The concept of zero-touch networks is centred on the idea of fully automated and self-managing networks that require minimal or no manual intervention from network administrators or other IT personnel. In the context of 5G networks and beyond, zero-touch networks are designed to be self-configuring, self-optimising, and self-healing, with the goal of reducing the need for manual intervention in network operations. To achieve a zero-touch network, 5G networks utilise advanced technologies such as machine learning, automation, and artificial intelligence. One important component of such technologies is the DE, which is a software system that uses these technologies to analyse network performance in real-time and make decisions about how to optimise the network. Together with the orchestrator, a DE can continuously monitor the network, identify and diagnose problems, and trigger corrective actions to maintain the desired level of network performance.

For the intelligent deployment and placement of containerised applications, including NFs, we rely in this work on a DE that incorporates multi-objective optimization algorithms (e.g., [35, 36]). Triggered by an orchestrator, the DE can run proactively or reactively based on the received monitoring input or orchestration requests, in order to allocate applications or NFs in the edge or cloud nodes. The DE has been designed to provide fast execution and efficient resource usage while scaling with the requests. Furthermore, a primary concern during its design was to build a system that could be easily extendable with new resource allocation algorithms.

The design of the DE (see Figure 5) is based on a controller instance and multiple workers. Each worker is composed of the Execution Engine and the library of the decision algorithms. This approach ensures that the DE will be able to quickly handle multiple execution requests, even in very complex infrastructures. The DE interoperates with the infrastructure telemetry services described in the previous sections in order to receive monitoring information about, for example, the available computing (CPU) and networking resources, but also additional information that may be deemed interesting by the network operator (e.g., aspects related to cost). With this information, the DE implements the decide part of the envisioned closed-loop control based on the principles of observe, decide, and act. The DE operates in a

FIGURE 5 Decision Engine design and main components.



technology-agnostic manner (e.g., in terms of virtual machines, containers, network functions, etc.), formulating the respective optimization problem in a generic way that involves workloads that need to be allocated in a converged computing, networking infrastructure.

The DE controller consists of the Access Interface and the Dispatcher. The former provides loose coupling between the DE and other systems like Prometheus and Kubernetes, exposing the necessary REST-based interfaces that allow bidirectional communication for exchanging commands, information, and notifications. In this regard, any orchestrator (e.g., [26]) can easily interact with the DE through the Access Interface. On the other hand, the Dispatcher is invoked by the Access Interface and manages the execution of the requests for executing resource allocation computations and handles the interaction with the multiple instances of the Execution Engine. The Execution Engine receives instructions from the Dispatcher for starting or terminating algorithm executions, and performs all the required actions, including the preparation of the execution environment, the monitoring of the execution progress and the handling of the final results or possible failures. The resource allocation algorithms can be implemented as plug-ins to the DE, exposing a common interface, independent of the implementation technology and the algorithms' internal logic, which among others determine the explicit syntax of the input parameters and the results for all algorithms. JSON is being used as the data-interchange format. Moreover, the controller service is able to automatically scale up or down the number of the available execution engines with respect to the number and complexity of the submitted execution requests.

The DE is implemented in Python as two different services (i.e., controller and execution engine) following the cloud native approach. To this end, there are available the appropriate container images for each service along with the required configuration (i.e., ConfigMap object) and deployment (Deployment and Service Objects) descriptions that enable their transparent deployment into any Kubernetes clusters. The configuration includes, among others, the minimum number of execution engines that should be always available.

6 | USE CASES EVALUATION

This section showcases and validates the presented end-to-end monitoring framework featuring a DE instance by displaying different metrics, including both infrastructure and RAN parameters. It does so through three use case experiments. The first experiment deals with UPF re-selection in edge-enabled 5G networks, the second focuses on RAN (gNB) measurements under UE mobility, and the third demonstrates containerised application placement and relocation via the DE.

6.1 | UPF re-selection towards the edge

As for Experiment #1, we trigger the following series of events: first (1), the 5GC CNFs are deployed through the installation of Helm charts, and the session management function (SMF) assigns both UEs to the core UPF. Second (2), UE1 starts a 100 Mbps UDP downlink iperf connection from an iperf server located in the core node. Then (3), UE2 initiates an iperf of the same characteristics until (4), where the iperf stops and the SMF re-assigns UE2 to the MEC UPF.[4] A new iperf connection is then started by UE2 pointing to the edge node iperf server in (5) until (6), the moment at which both UEs stop their corresponding iperf connections. Finally, the whole deployment (5GC CNFs) is terminated, and the Open5GS containers are deleted in the core and edge nodes. The considered events are listed in Table 1.

Figure 6 shows the Grafana dashboard used for Experiment #1 consisting of three panels: two for infrastructure metrics (node CPU and networking transmit) measured at the core and edge nodes, and one for a RAN metric (downlink bitrate) measured at the Amarisoft Callbox acting as gNB. The temporal events are highlighted with red circles. In (1), we

[4]When a UE initially attaches to the network, the SMF assigns a default PDU Session targeting the core UPF to ensure basic connectivity for the UE. Then, to utilize a second UPF located at the network edge, a dedicated PDN connection from the UE to this specific UPF must be established. To accomplish this, we employ the pdn_connect command provided by Amarisoft.

observe a peak of CPU caused by the deployment of the 5GC CNFs in the nodes. In (2), CPU and network transmit increase due to the iperf traffic triggered by UE1. Then, in (3), UE2's traffic raises the CPU and transmit networking of the core node, since UE2 is also assigned to the core UPF. Instead, when UE2 is assigned to the MEC UPF (5), CPU and networking resources are shared between the core and edge node. Finally, we observe a peak on CPU in (7) corresponding to the CNFs termination.

This use case shows the potential value of the presented monitoring framework in 5G deployments, where important metrics from different domains (e.g., infrastructure and RAN) can be assessed in an integrated and automated end-to-end manner.

TABLE 1 List of events in experiments #1 (UPF re-selection) and #2 (UE mobility).

Event	Description
#1.1	5GC CNFs deployed and UEs assigned to core UPF
#1.2	UE1 starts a 100 Mbps UDP downlink iperf connection
#1.3	UE2 starts a 100 Mbps UDP downlink iperf connection
#1.4	UE2 iperf stopped and assigned to MEC UPF
#1.5	UE2 restarts a 100 Mbps UDP downlink iperf connection
#1.6	Both UEs stop their corresponding iperf connections
#1.7	Whole deployment (5GC CNFs) is terminated
#2.1	UE1 starts a 120 Mbps uplink iperf to the core node
#2.2	gNB reduces the receiver gain 4 dB (−4 dB aggregated)
#2.3	gNB reduces the receiver gain 4 dB (−8 dB aggregated)
#2.4	gNB reduces the receiver gain 4 dB (−12 dB aggregated)

6.2 | UE mobility

Finally, to test the containerised Amarisoft sampling function, in Experiment #2 we focus solely on RAN metrics. To do so, we show in Figure 7 a Grafana dashboard corresponding to a scenario where UE1 moves away from the gNB, resulting in higher path loss (lower SNR) and lower MCS and, consequently, lower bit rates. In particular, the series of events (see Table 1) is as follows: (1) UE1 starts a 120 Mbps uplink iperf connection to the core node, and from (2) to (4) we sequentially reduce by 4 dB the receiver gain at the gNB through the Amarisoft API to emulate UE1 moving away. As expected, this results in higher path loss and lower SNR at the gNB, achieving lower MCS and lower bit rate.

FIGURE 6 Grafana dashboard for experiment #1 (UPF re-selection). Events are numerated within red circles.

FIGURE 7 Grafana dashboard for Experiment #2 (UE mobility). Events are numerated within red circles.

Experiment #2 has therefore demonstrated the suitability of the developed Amarisoft sampling function in terms of integrating pure RAN metrics into the whole monitoring framework. Providing such end-to-end monitoring is of critical importance to let 5G network administrators control and manage their services, especially when it comes to end-to-end network slicing.

6.3 | Containerised application relocation

As for the last use case, or Experiment #3, we next cover the scenario in Figure 8 to showcase the DE. Similarly to the previous scenarios, we deploy the 5G NFs in two nodes spanning the edge and cloud domains. However, we now provide more Kubernetes workers, so to habilitate a larger number of candidate nodes to run the application of interest, which can be regarded as a containerised VR/AR application. Consequently, we can focus on the decision-making pipeline, which will determine the optimal node where the application of interest should run.

As in ref. [26], the orchestrator, which is implemented as a Python script, uses the Kubernetes scheduler to execute its actions. In this work, however, the orchestrator relies on the DE to identify the optimal node where to deploy the containerised application of interest. Upon request reception, the orchestrator proceeds to the deployment and the continuous life-cycle management of the application. Then, the DE periodically identifies the most convenient node by estimating the value of each node according to the node CPU, the end-to-end latency between the application of interest and the node, and the cost of hosting the application of interest at each node. If the DE informs the orchestrator that the application needs to be moved from one node to another, it initiates the relocation process using the Kubernetes Python client.

The DE makes use of a multi-objective optimization mechanism based on refs. [35, 36] that consists of two phases: *i*) it first computes a set of candidate Pareto optimal resources that also have a sufficient computing capacity to serve the application, and then *ii*) it selects the resource that minimises a desired optimization function.

In particular, the mechanism included in the DE receives as input a set $\mathcal{S}$ of the available resources, including their characteristics in terms of available computing capacity p, access latency between the end-user and the resource l, and the cost of using the respective resource c. This way, each resource $\mathcal{R}$ is characterised by the tuple $\mathcal{R} = \{p, l, c\}$. In the first phase, the DE removes from set $\mathcal{S}$ resources whose available computing capacity p is smaller than the application's computing requirements p_a (i.e., $p_a > p$). If resources with sufficient capacity are not available, the demand is blocked. Otherwise, the DE serves the demand by initially finding the Pareto optimal set, discarding further resources dominated by others. According to the multi-objective optimization theory, we say that a resource $\mathcal{R}_A$, in our DE setting, is dominated by another resource $\mathcal{R}_B$, if the following holds: $p_A \leq p_B$, $l_A \geq l_B$, and $c_A \geq c_B$. In other words, resource $\mathcal{R}_B$ dominates resource $\mathcal{R}_A$ if $\mathcal{R}_B$ has larger available computing capacity, lower latency, and also results in a lower cost than $\mathcal{R}_A$.

In the second phase, among the candidate non-dominated resources that have been discovered, the DE selects the one that optimises a cost function. We use for this example a simple formulation that minimises $f = (w_1 p^{-1} + w_2 l + w_3 c)$, where w_i are weights depending upon the prioritisation configuration. Of course, more fine-grained optimization functions can be tailored according to the scenario.

The service request from the orchestrator to the DE comes through a REST API POST HTTP command, incorporating in the body of the HTTP message a JSON file (see Listing 1) including the monitored parameters in $\mathcal{R}$. In such an example, we observe that the latency at the cloud node currently hosting the application of interest (i.e., worker-4) is much higher than in the edge nodes (e.g., worker-2 and worker-3). Besides, we notice that worker-2 is heavily loaded in terms of CPU (i.e., p^{-1}) compared to worker-3. So, when the DE runs one of the available resource allocation algorithms (e.g., the ComputeNode as in the example), it identifies worker-3 as the optimal node according to its estimated value f. Then, it asynchronously responds to the orchestrator with a specific GET HTTP-based REST command, as shown in Listing 2, which upon receiving the response triggers a container relocation from worker-4 to worker-3 through the Kubernetes client.

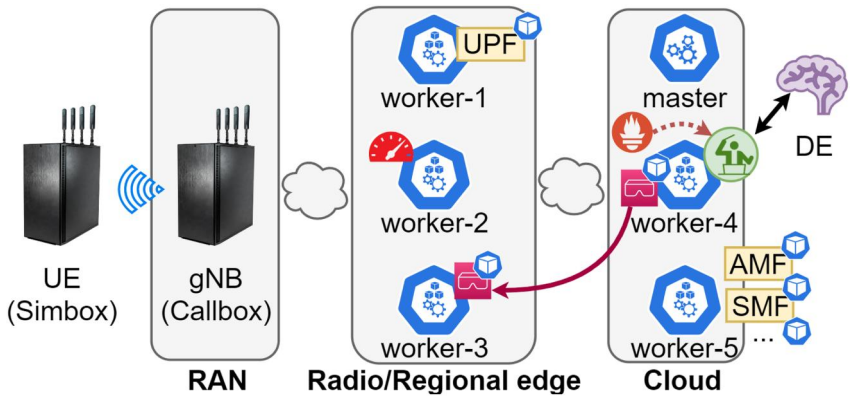


FIGURE 8 Scenario for the containerised application relocation.

```
{ "execution_plugin": "ComputeNode",
"parameters":
{"telemetry":
[{"name": "worker-2"
"cpu": "48.54",
"latency": "1.11",
"cost": "4"},
{"name": "worker-3"
"cpu": "8.29",
"latency": "1.08",
"cost": "4"},
"name": "worker-4"
"cpu": "6.85",
"latency": "20.07",
"cost": "1"},
...]
}
}
```

Listing 1: Monitoring data input sent by the orchestrator to the DE.

```
{
"created_at": 1669713816,
"engine_id": "268be485-2a00-418b-952b",
"execution_id": "8c6bb879-4425-4b10-8c3b,
"result": "{'compute_node3': 'worker-
3'}",
"status": 2,
"updated_at": 1669713816
}
```

Listing 2: Output of the DE decision. In this case, node worker-3 is selected for running the containerised application of interest.

With theuse of this case, we demonstrate the effectiveness of using the Kubernetes scheduler altogether with a DE for achieving zero-touch orchestration, enabling automated updates and maintenance without any manual intervention. In this regard, notice that the containerised application of interest runs transparently to the user, without any interruptions or downtime, even when updates or changes are made to the underlying infrastructure since the Kubernetes scheduler ensures that a new pod (with the containerised application of interest inside) is created before the previous one is deleted, resulting in a seamless transition for the application.

7 | CONCLUSIONS

In this work, we developed and evaluated a cloud-native 5G framework with containerised end-to-end monitoring. Using a 5G testbed with over-the-air transmissions, we demonstrated how to integrate a fully functional 5G framework using containerised network functions in a multi-node Kubernetes cluster, including an open-source 5G core, a commercial radio access network, and an end-to-end monitoring system. We illustrated the capabilities of the framework through Grafana dashboards that show various metrics from both the infrastructure and radio/RAN domains. Additionally, we added intelligence to the experimental platform through a decision engine that interacts with the Kubernetes scheduler through a zero-touch orchestrator. Furthermore, as part of our ongoing research, we plan to extend our experimentation by incorporating real UEs to validate the performance and functionality of our proposed system in real-world scenarios.

AUTHOR CONTRIBUTIONS

Sergio Barrachina-Muñoz: Conceptualization; investigation; methodology; software; supervision; validation; visualization; writing—original draft; writing—review and editing. **Rasoul Nikbakht**: Conceptualization; methodology; writing—review and editing. **Jorge Baranda**: Conceptualization; methodology; validation; writing—review and editing. **Miquel Payaró**: Conceptualisation; methodology; project administration; writing—review and editing. **Josep Mangues-Bafalluy**: Conceptualization; project administration; supervision; writing—review and editing. **Panagiotis Kokkinos**: Conceptualization; software; writing—review and editing. **Polyzois Soumplis**: Conceptualization; software; writing—review and editing. **Aristotelis Kretsis**: Conceptualization; software; writing—review and editing. **Emmanouel Varvarigos**: Conceptualization; software; writing—review and editing.

ACKNOWLEDGEMENTS

This work has partially been funded by the MARSAL project from EU Horizon 2020 research and innovation programme under grant agreement No 101017171 and from the Spanish MINECO grant TSI-063000-2021-56/TSI-063000-2021-57 (6G-BLUR).

CONFLICT OF INTEREST STATEMENT

None.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in Zenodo at https://doi.org/10.5281/zenodo.6075481, reference number 10.5281/zenodo.6075481.

ORCID

Sergio Barrachina-Muñoz https://orcid.org/0000-0002-7941-1018

Josep Mangues-Bafalluy https://orcid.org/0000-0003-4960-9434

REFERENCES

1. Chochliouros, I.P., et al.: Machine learning-based, networking and computing infrastructure resource management. In: IFIP International Conference on Artificial Intelligence Applications and Innovations, pp. 85–94. Springer (2021)
2. Vardakas, J.S., et al.: Towards machine-learning-based 5g and beyond intelligent networks: the marsal project vision. In: 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), pp. 488–493. IEEE (2021)
3. Mekki, M., Arora, S., Ksentini, A.: A scalable monitoring framework for network slicing in 5G and beyond mobile networks. In: IEEE Transactions on Network and Service Management (2021)

4. Novaes, C., et al.: Virtualized c-ran orchestration with docker, kubernetes and openairinterface. In: XXXVII Simposio Brasileiro de Telecomunicacoes e Processamento de Sinais (2020)
5. Haavisto, J., et al.: Open-source RANs in practice: an over-the-air deployment for 5G MEC. In: 2019 European Conference on Networks and Communications (EuCNC), pp. 495–500. IEEE (2019)
6. Arouk, O., Nikaein, N.: Kube5G: a cloud-native 5G service platform. In: GLOBECOM 2020-2020 IEEE Global Communications Conference, pp. 1–6. IEEE (2020)
7. Arouk, O., Nikaein, N.: 5G cloud-native: network management and automation. In: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–2. IEEE (2020)
8. Osmani, L., et al.: Multi-cloud connectivity for kubernetes in 5g networks. IEEE Commun. Mag. 59(10), 42–47 (2021). https://doi.org/10.1109/mcom.110.2100124
9. Trakadas, P., et al.: Scalable monitoring for multiple virtualized infrastructures for 5g services. In: SoftNetworking 2018, the International Symposium on Advances in Software Defined Networking and Network Functions Virtualization, pp. 1–4 (2018)
10. Giannopoulos, D., et al.: Monitoring as a service over a 5g network slice. In: 2021 Joint European Conference on Networks and Communications and 6G Summit (EuCNC/6G Summit), pp. 329–334. IEEE (2021)
11. Giannopoulos, D., et al.: A holistic approach for 5g network slice monitoring. In: 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), pp. 240–245. IEEE (2021)
12. Goshi, E., et al.: Investigating inter-nf dependencies in cloud-native 5g core networks. In: 2021 17th International Conference on Network and Service Management (CNSM), pp. 370–374. IEEE (2021)
13. Li, X., et al.: 5growth: an end-to-end service platform for automated deployment and management of vertical services over 5g networks. IEEE Commun. Mag. 59(3), 84–90 (2021). https://doi.org/10.1109/mcom.001.2000730
14. Li, X., et al.: Topology-aware resource allocation for iot services in clouds. IEEE Access 6, 77880–77889 (2018). https://doi.org/10.1109/access.2018.2884251
15. Kiran, N., et al.: Vnf placement and resource allocation in sdn/nfv-enabled mec networks. In: 2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), pp. 1–6 (2020)
16. Pallewatta, S., Kostakos, V., Buyya, R.: Microservices-based iot application placement within heterogeneous and resource constrained fog computing environments. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, pp. 71–81 (2019)
17. Barrachina-Muñoz, S., Payaró, M., Mangues-Bafalluy, J.: Cloud-native 5G experimental platform with over-the-air transmissions and end-to-end monitoring. In: CSNDSP 2022, Porto, pp. 22–26. IEEE (2022)
18. Whitepaper: The Status of Open Source for 5G, 5G Americas, Tech. Rep. MSU-CSE-06-2, 2019. [Online]. https://www.5gamericas.org/the-status-of-open-source-for-5g/
19. 3GPP: System architecture for the 5G system (5GS)," 3rd generation partnership project (3GPP). Technical Specification (TS) (2021)
20. Pateromichelakis, E., et al.: End-to-end data analytics framework for 5G architecture. IEEE Access 7, 40295–40312 (2019). https://doi.org/10.1109/access.2019.2902984
21. Nikaein, N., et al.: OpenAirInterface: a flexible platform for 5G research. Comput. Commun. Rev. 44(5), 33–38 (2014). https://doi.org/10.1145/2677046.2677053
22. Free5GC. https://www.free5gc.org/. accessed: 2023-02-07
23. Open5GS. https://open5gs.org/. accessed: 2023-02-07
24. Chun, B., et al.: Kubernetes enhancement for 5G NFV infrastructure. In: 2019 International Conference on Information and Communication Technology Convergence (ICTC), pp. 1327–1329. IEEE (2019)
25. Use Containers to Build, Share and Run Your Applications, https://www.docker.com/resources/what-container, accessed: 2023-02-07
26. Barrachina-Muñoz, S., et al.: Intent-based orchestration for application relocation in a 5G cloud-native platform. In: 2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 94–95. IEEE (2022)
27. What Is Calico," https://projectcalico.docs.tigera.io/about/about-calico, accessed: 2023-02-07
28. UERANSIM GitHub repository. https://github.com/aligungr/UERANSIM. accessed: 2023-02-08
29. AMARI Callbox Ultimate," https://www.amarisoft.com/app/uploads/2022/01/AMARI-Callbox-Ultimate.pdf, accessed: 2023-02-08
30. AMARI UE Simbox Series. https://www.amarisoft.com/app/uploads/2021/12/AMARI-UE-Simbox-E-Series.pdf. accessed: 2023-02-08
31. Nikbakht, R., et al.: Mobile edge vertical applications using ETSI MEC APIs and sandbox. In: IEEE Conference on Standards for Communications and Networking (CSCN). IEEE (2022)
32. Kube-prometheus GitHub Repository, https://github.com/prometheus-operator/kube-prometheus, accessed: 2023-02-08
33. Rabenstein, B., Volz, J.: Prometheus: A Next-Generation Monitoring System (Talk). USENIX Association, Dublin (2015)
34. Tseng, Y., et al.: Re-think monitoring services for 5G network: challenges and perspectives. In: 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 34–39. IEEE (2019)
35. Soumplis, P., et al.: Resource Allocation Challenges in the Cloud and Edge Continuum, pp. 443–464 (2022)
36. Kontodimas, K., et al.: Secure distributed storage on cloud-edge infrastructures. In: 2021 IEEE 10th International Conference on Cloud Networking (CloudNet), pp. 127–132 (2021)

How to cite this article: Barrachina-Muñoz, S., et al.: Deploying cloud-native experimental platforms for zero-touch management 5G and beyond networks. IET Netw. 1–11 (2023). https://doi.org/10.1049/ntw2.12095