Design and Validation of an Open Source Cloud Native Mobile Network

Nikolaos Apostolakis, Marco Gramaglia, and Pablo Serrano

***Abstract*—Network technologies are embracing the *cloud-native* paradigm, following the current best practices in cloud computing. Cloud-native technologies might be applied to different types of network functions in a mobile network, but they are particularly relevant nowadays for core network functions, as the recent standard introduces the Service-Based Architecture that matches modern *cloud-native* technologies such as Docker or Kubernetes. In parallel, a number of open source software initiatives already provide researchers and practitioners with usable software that implements the key functionality of a mobile network (both for LTE and 5G). These software solutions, however, are monolithic and not integrated into state-of-the-art cloud-native frameworks. In this paper, we fill this gap by describing the implementation of a cloud-native mobile network, which supports channel emulation and provides an affordable and scalable way of testing orchestration algorithms with standardized VIM interfaces. Our experimental evaluation shows the applicability of our solution, which is released as open-source and illustrates its flexibility.**

***Index Terms*—Cloud-native system, Mobile Network, Open-Source.**

I. Introduction

FOLLOWING the recent needs for higher levels of network flexibility, programmability, and automation, the Network Function Virtualization (NFV) [1] technology can be considered as one of the most important drivers for the 5G networks (and beyond). When compared to (legacy) 4G/LTE services, 5G network services exhibit very diverse characteristics, e.g., they may operate in very short timescales, should be restricted to certain geographic locations, may require more integrated means of interacting with the network, and eventually may require higher degrees of flexibility. Because of this, proprietary hardware and telecommunication protocols, especially the ones related to network management, have often been considered a severe impediment to the development of novel technologies.

The characteristics of NFV cope well with the increasingly demanding requirements of 5G Networks and beyond. Besides the advantages in terms of operational costs, due to the commoditization of the network through its softwarization, the extreme flexibility of this paradigm for (re-)configuration, scaling, and continuous deployment and integration matches the very diverse 5G network stakeholder ecosystem. With this new approach, it is possible to deploy services on commodity off-the-shelf servers (COTS), traditionally used for cloud computing. This has enabled communication providers to multiplex services over the same physical infrastructure, share resources between the different functions, deploy on-demand per user load and, consequently, reduce operational expenses (OPEX).

However, even though virtualization technologies based on virtual machines provide full hardware independence, they are based on hypervisors, an intermediate virtualization layer that is considered difficult to operate and manage, and that may include unacceptable loads when dealing with fast timescale operations. Also, the deployment, configuration, and management of novel networking services are prone to a series of drawbacks, resulting from the tight dependency between software and hardware. This constitutes an obstacle to the effective deployment of the NFV technology and has motivated significant efforts to migrate from these "static" physical network functions (PNFs) to "dynamic" network functions (VNFs). As a result, the operators have been reluctant to discard their investments in PNFs for a solution that struggles to yield the desired levels of resiliency [2].

To overcome the above issue, cloud network functions (CNF) have gained a lot of attention, as they are already widely used in cloud computing environments, thus tested for their robustness. CNFs rely on a very thin virtualization layer, and have a minimal footprint on the infrastructure, allowing to implement modern software architectures such as microservices [3], and providing a very high degree of configurability. In fact, CNF is attracting a lot of interest: on the one hand, standardization bodies such as ETSI have adopted orchestration frameworks tailored to this paradigm [4], while on the other hand, the open source Cloud Native Computing Foundation (CNCF) [5] has already defined a roadmap composed by technologies, tools, and patterns for Contenarization, CI/CD, Observability, Messaging, Distributed Databases and, most importantly, Networking. However, as CNFs are just arriving in the telecommunications domain, they are lacking the required maturity levels prior to their adoption in production environments (beyond the proprietary solutions). Since the use of CNFs in mobile networking requires overcoming many challenges [6], a *democratization* of the corresponding software components (e.g. released as open source) shall foster the development and testing of the required solutions.

Motivated by the above, in this paper we present an end-to-end mobile network with channel emulation capabilities, monitored and managed by a CNF orchestrator, which is released as open-source. Our contributions are:

N. Apostolakis is with the IMDEA Networks Institute, 28918 Leganés, Spain, and also with the Telematic Engineering Department, University Carlos III of Madrid, 28911 Leganés, Spain (e-mail: nikolaos.apostolakis@imdea.org).

M. Gramaglia and P. Serrano are with the Telematic Engineering Department, University Carlos III of Madrid, 28911 Leganés, Spain (e-mail: {mgramagl,pablo}@it.uc3m.es).

- We provide researchers and practitioners with an open-source scalable solution to emulate a complete 5G mobile network, building on well-known cloud-native tools.
- We develop the required modules and tools to support the network operation, namely: (*i*) a module to support the channel emulation, which is particularly useful for those scenarios where specialized hardware is not available, or in repeatable experiments with varying channel conditions are to be performed; (*ii*) data shippers to monitor the different network functions, supporting the implementation of autonomous network management decisions; and (*iii*) a collection of configuration blueprints (i.e., Helm charts) to automatize the orchestration processes.
- We develop two use cases to illustrate the features of our solution and to validate the behavior of the tools and modules developed.

As mentioned, the interested practitioners can experiment with our proposed cloud-native mobile network, as it is available in open source (including the code required to deploy the use cases). For consistency reasons we split the codebase into core and orchestration (available at https://github.com/kaposnick/k8s-open5gs), and the access (available at https://github.com/kaposnick/srslte).

The rest of the paper is structured as follows: Section II presents the design of the solution, introducing the different building blocks required to build a cloud-native mobile network; Section III describes the procedures to deploy the network; Section IV introduces the experiments we performed to benchmark the solution and assess its performance; Section V discusses related research work and initiatives; finally, Section VI concludes the paper.

II. Design of a Cloud-Native ecosystem

The softwarization of mobile network functions allowed network operators to reduce their operational expenditure by using open software and also enabled researchers and practitioners to experiment with new network functionalities. This software-driven transition is being accelerated with the introduction of cloud-native architectures [7], which greatly increases the flexibility and scalability of the software. In what follows, we introduce the different components of our cloud-native mobile network design, several of them chosen among the set of open-source solutions that have become available during the last few years (we review them in Section V).

A. Overview

The different components of our design are illustrated in Figure 1. For orchestration, we chose Kubernetes, an open-source system for deploying and managing containerized applications. For the core network implementation, we selected Open5Gs, motivated by its adaptability to the cloud native paradigm. For the UE and the Access network, we rely on srsRAN, an open-source software that provides a full mobile network stack implementation for those elements. Furthermore, srsRAN community is very active and its contributors keep intensively improving the product to keep up with the latest 3GPP standards. To simplify the deployment and

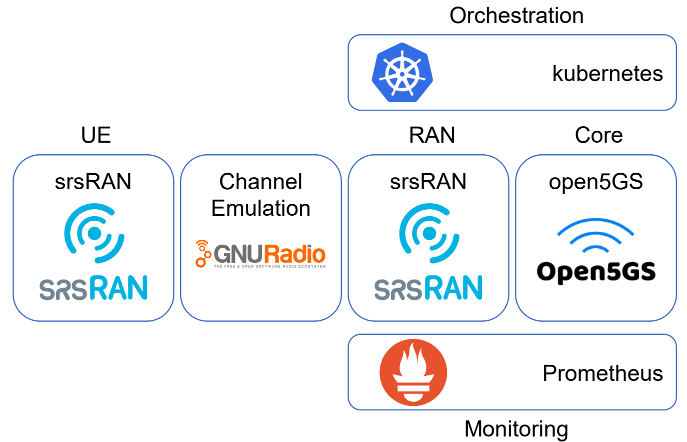

Fig. 1: The solutions that build the end-to-end architecture presented in this paper.

support repeatability, we developed a simple channel emulator, which enables deploying a fully programmable end-to-end network without requiring expensive hardware. Finally, we chose Prometheus, one of the monitoring solutions promoted by the CNCF [5], as the monitoring framework to collect the status of the whole system. In what follows, we detail these components.

B. Management and Orchestration

Kubernetes (also known as `k8s`) is a container orchestration platform, widely adopted in the cloud computing domain for more than five years. A typical cloud native application consists of a set of container images (e.g. Docker images) that are deployed and managed by Kubernetes, using configuration files that describe the functionality and the services they provide. Its scaling and redundancy capabilities have already been demonstrated in production environments, and it is envisioned that telco operators will rely on Kubernetes for next-generation networks, motivated by the need to make more efficient use of the heterogeneous virtualized infrastructure (which spans over central and edge data centers). To manage the additional management complexity, a huge ecosystem of open source projects that extend the possible operational tasks has been created under the umbrella of the Cloud Native Computing Foundation (CNCF [5]).

C. Core Network functions

Open5Gs is a very popular open-source implementation of a mobile network core. Written in C, it stands as a reference among researchers and mobile telecommunications practitioners for experimentation and future enhancements. Currently supporting up to 3GPP 5G Release 16, it contains the most important components of the 5G Core (5GC) and 4G Evolved Packet Core (EPC) with Control-User Plane Separation (CUPS) [8], meaning it can operate on both 5G Non-Standalone (NSA) and Standalone (SA) modes, as it can serve both 4G E-UTRAN NodeB (eNBs) and 5G next Generation NodeB (gNBs). Its straightforward build procedure

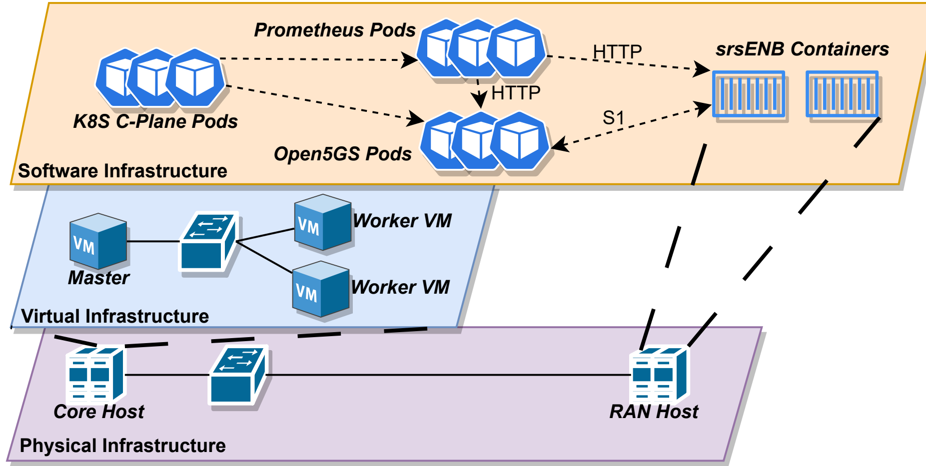

Fig. 2: Core and RAN Infrastructure setup.

makes its deployment in small-scale private networks very easy, while its modular architecture could be considered as a natural candidate for running it into microservices-based cloud-native environments powered by Kubernetes, as we are going to focus in this paper. As illustrated in Fig. 2, Open5Gs *pods* will be deployed over the virtual infrastructure, while srsENB is directly running on top of Docker containers.

D. RAN functions

srsRAN [9], formerly srsLTE, is another open-source software framework, developed by Software Radio Systems, that provides the functionality from the PHY up to RRC layer for eNB/gNBs, while also supporting 4G/5G UEs. It is written in C/C++, and its configuration parameters cover a wide range of base station and UE possible configurations. Regarding the RF interface, its contributors have developed drivers for various commercial hardware RF-frontends like URSP, Soapy SDR and BladeRF.

Additionally, srsRAN provides a software RF-frontend based on ZeroMQ, an open source message queueing library written in C. When using this driver, the transmitted I/Q baseband symbols between UE and base station are transferred over various transport methods, like inter-process communication (IPC) or TCP sockets. Choosing this driver avoids the need for high expertise in RF channel configuration and facilitates the introduction of researchers who want to simulate a radio access network environment, but whose RF channel is not their main area of interest or would be reluctant to invest in actual hardware transceivers. We next discuss how we take advantage of ZeroMQ to implement a channel emulation model.

E. Channel emulation: the GRC broker

As mentioned, the use of ZeroMQ enables running the network in a fully softwarized way. Furthermore, it also enables the emulation of complex topologies via programming (e.g., arbitrary complex topologies can be created by dynamically connecting endpoints, as they do in large-scale emulators using hardware in the loop [10]). The main challenge when emulating complex mobility scenarios relies on the handling of I/Q symbols, e.g., sending low power symbols from the eNB to trigger a Handover Request from the UE.

Following the srsRAN Documentation, we utilized the GNU Radio Companion (GRC), which is another open source project mainly used for software-defined radio (SDR), and, among others, contains modules for the ZeroMQ library. We coded in Python a `GRC Broker`, located between the UE and the cells of the eNB(s) implementing the RF interface, as illustrated in Fig. 3. This module intercepts the transmitted I/Q symbols and performs operations on them to emulate the channel condition for the two traffic directions:

- In the downlink direction, in order to simulate the distance between the UE and the cells, we made use of multiplier blocks followed by an adder. Multiplier blocks multiply the time-domain cells' transmitted samples with a constant gain value in the range $[0, 1]$, adjusting UE's perception of the cells' signal strength. The adder block simply performs the superposition of the modified samples, as would be the case in the real environment.
- In the uplink direction, we used a throttle block that retransmits the I/Q symbols towards the cells of the eNB(s).

We leverage this setup to create different evaluation scenarios (discussed in Section IV) and leave as future work the implementation of additional features into the GRC Broker.

F. Monitoring

In order to have a consistent view of the mobile network's current status in a unified platform, we deployed the cloud-native version of Prometheus (*kube-prometheus-stack*) in the same Kubernetes cluster used for the Open5Gs. In this section, we describe the steps taken in order to monitor both Core and RAN through Prometheus.

1) Core Network: When *kube-prometheus-stack* is deployed in a Kubernetes cluster, by default it installs exporters in all the master and worker nodes, which collect and expose metrics like CPU usage, RAM, Networking status, and IOPS on different granularity levels (node, pod, container).

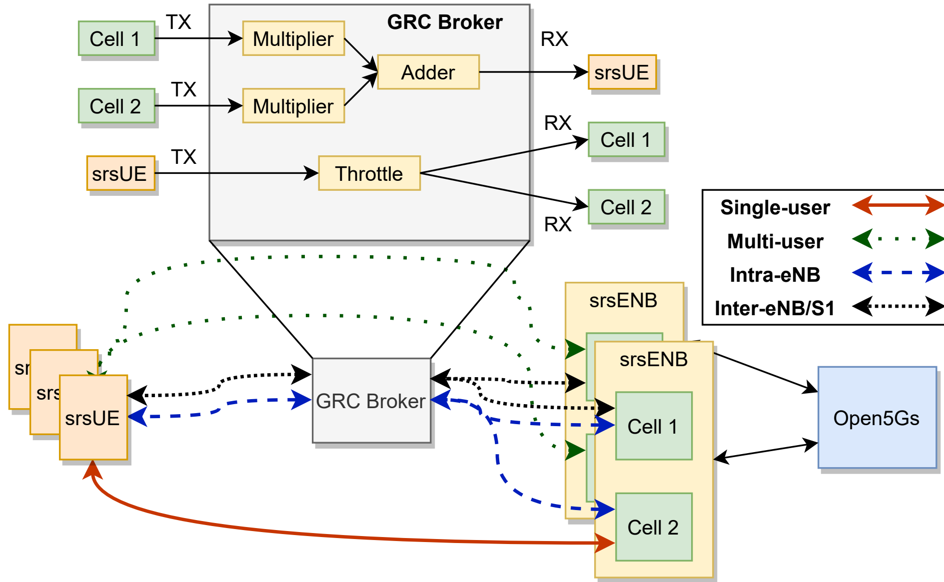

Fig. 3: The evaluated scenarios.

By exporting those metrics we can monitor the health of the virtual infrastructure running the different core network functions, as depicted in Figure 2. Prometheus operator pods have set those exporters as targets and will start polling them in fixed intervals using HTTP in a well-known path: (`/metrics`). Those metrics are directly accessible either through the Prometheus WebUI or Grafana, a visualization tool, using PromQL, a functional query language.

2) RAN: srsRAN produces an array of metrics for the lower layers of the networking stack which can be exported to the console or to a text file. However, srsRAN does not have an agent acting as a metrics exporter toward 3rd party software. Therefore, we modified the source code of the srsENB in order to develop and integrate a new one into the final binary. The first step towards this direction was to integrate *lighttpd* project, a lightweight open-source HTTP server written in C programming language. Then, we developed the REST API through which the metrics, in key-value pair format, will be exported to the Prometheus operator. Afterward, we developed Kubernetes manifests which comprise a pod, whose role is to proxy the HTTP requests, received by the Prometheus operator, towards the end srsENB process. Those manifests are deployed to the Kubernetes cluster with the IP address of the srsENB as an input parameter. Last, facilitating the discovery capabilities of Prometheus, we deployed a *Service-Monitor* configuration resource, which automatically discovers the 'proxy' pods existing in the cluster and sets them as Prometheus targets.

Thus, for every srsENB process we instantiate, an auto-discovery service is performed and, once it is completed a few seconds later, the eNB starts getting polled (or scraped, using the Prometheus terminology).

III. Building and Deploying the Network

One of the main advantages of cloud-native solutions is their extreme flexibility thanks to the easy templating and parametrization of the software components. In the following, we discuss how we use two key components of our setup, manifests and Helm charts, to perform the mobile network configuration.

A. Kubernetes Manifests

To be able to manage very heterogeneous software components, Kubernetes needs to abstract the specific parameters associated with them. To this aim, the so-called *Manifests* are used to define the information model (i.e., the parameters associated with each function) for each network function. Thus, we had to provide the Kubernetes resource files for each software component in the network architecture. Open5Gs provides software implementing the Network Functions (NFs) for both EPC and 5GC under the same umbrella, as depicted in Figure 4:

- *Control-plane functions*: MME, HSS, PCRF, SGW-C, PGW-C (for the 4G/LTE); AMF, SMF, UDM, UDR, NRF, NSSF, AUSF, PCF, BSF (for the 5GC).
- *User-plane functions*: SGW-U, PGW-U (for the 4G/LTE), and UPF (for the 5GC).

Hence, manifests include information such as the exposed ports, the internal interconnections, and the software components to be deployed by Kubernetes. Each NF is instantiated within the context of a *StatefulSet* controller, instead of a standalone *Pod*, so that it is appropriately managed in case of termination. Additionally, to support realistic use-case scenarios, where control plane and user plane NFs should be closely located but in distinct nodes, we set the *nodeSelector* attribute to `mobile-core: control` for control plane NFs and `mobile-core: user` for user plane NFs. This attribute is enforced by Kubernetes at the scheduling stage.

For the inter-pod communication, we use the *ClusterIP* service type, so that the corresponding service pods are accessible only within the Kubernetes cluster (preventing external

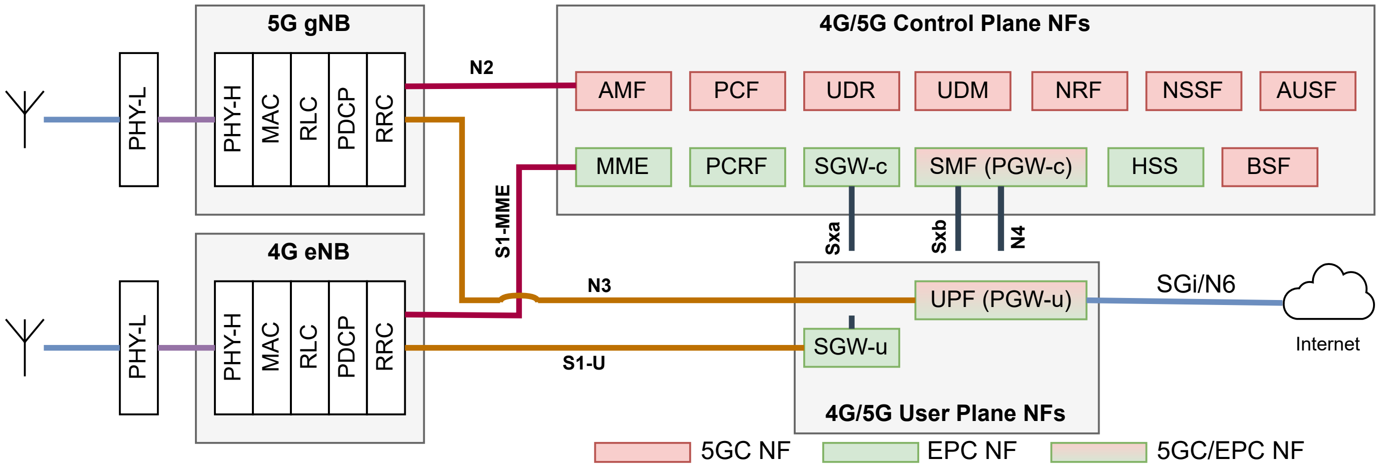

Fig. 4: The network functions included in Open5Gs and srsRAN.

access), while for the interfaces exposed towards the RAN (i.e., S1-MME/N2 by the MME/AMF, S1-U/N3 by the SGW-U/UPF) we use the *LoadBalancer* service type. This type of service is used in cloud environments to enable the external connectivity towards the cluster's pods, and typically uses a load balancer at the frontier of the cloud DC.

B. Helm Charts

The deployment stage of a cloud computing service (a mobile network core, in this case) has proven to be prone enough to human errors for a number of reasons:

- A considerable amount of Kubernetes manifests ($\sim$ 3 manifests per NF) has to be applied for every single iteration of an experiment. During the de-provisioning of the service, the reverse procedure has to be executed taking care to de-instantiate all the network functions.
- A few manifests attributes have to be provided at the initial stage, as they depend on the current state of the environment. Such attributes are the IP addresses that must be accessible by the RAN. Statically defining them, such as in a production environment like in an operator's DC is a solution that limits the flexibility of the deployment in a local environment, as they depend on the Network Interface Card (NIC) IPs, which are obtained through DHCP and are renewed upon reboot. Those IPs are used for setting the *LoadBalancer* service configuration as described above and the IP address that the UPF advertises to the SMF (and consequently to the AMF and the gNB) when addressing a service request.

Thus, as already done by major cloud computing services, we employ Helm charts to automate those configuration tasks by storing parameterized Kubernetes manifests along with a single configuration file (containing all the global parameters) in a centralized location (this chart can be uploaded to an online registry for accessibility reasons, as for the case of the Docker images). Then Helm automatically generates the equivalent Kubernetes manifests and afterward provides them to the Kubernetes API server. This results in a single command executed for (de)provisioning of the entire core network, thus greatly improving the automation.

IV. Experiments

We ran the Kubernetes cluster on a KVM cloud platform using Ubuntu 18.04.6. We then used Ubuntu 20.04.1 LTS as the image of all the nodes of the cluster (one master and two workers, as illustrated in Fig. 2), with Docker 20.10.8 as the container runtime, as well as `kubelet` and `kubectl`. The control plane was set using the `kubeadm` utility and Flannel as the Container Networking Interface (CNI). The network functions were connected to an internal network for control plane and inter-pod communication, and NAT forwarding is enabled for management purposes through SSH. The two worker nodes had an additional network adapter of type `Bridged Adapter` attached to another Ethernet Adapter of the host machine, exposing the N2 and N3 interfaces towards the gNB(s). The two worker nodes were also annotated with the labels `mobile-core:control` and `mobile-core:user`, which are taken into account during the deployment stage of the CNFs. The RAN (srsENB/srsUE) is set up using Docker in a commercial laptop with the 4 cores CPUs at 1.8 Ghz and 16 GB of RAM. Each srsUE process runs in a separate networking namespace in order to have distinct routing tables.

In what follows, we deploy two different scenarios to illustrate the use of our platform and validate the adequate behavior of the developed modules. For each considered scenario, the Open5Gs helm chart had to be deployed from scratch to fully decommission software components from previous runs. Also, the UDR database had to be populated with UEs information for authentication purposes. At each chart deployment, the core network reached a healthy state after about approx. 2 minutes, with several pods restarts due to unfulfilled interdependencies: for instance, the UDM container has to wait for the UDR database container to start up, otherwise, it is restarted. To check the end-to-end state of the core network, we access the Prometheus REST API (which exposes the retrieved metrics, as discussed before). We

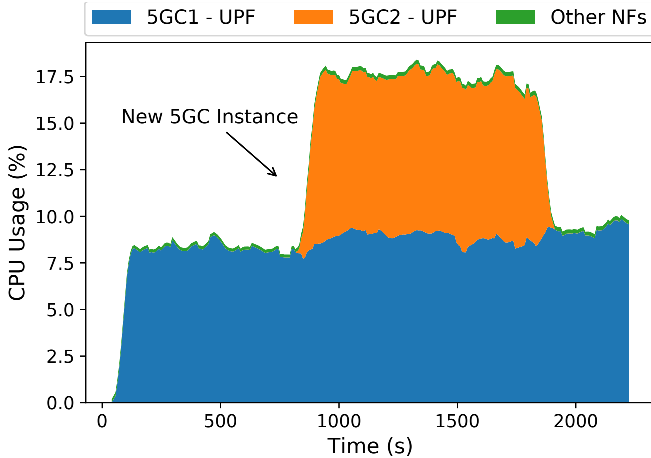

Fig. 5: CPU Utilization of two 5G Core instances.

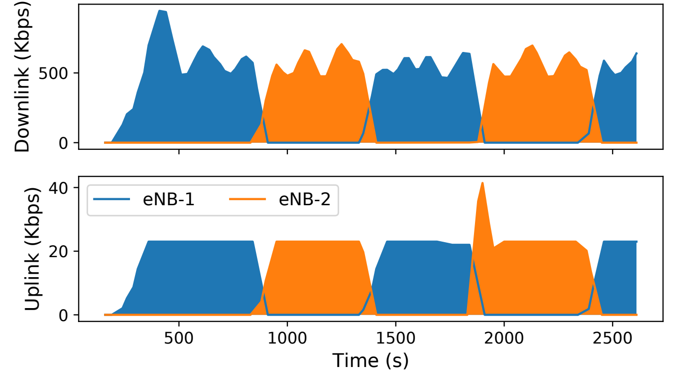

Fig. 6: RAN throughput.

then analyze the performance of the system for the following scenarios. We remark that, in addition to these two scenarios, the platform can be used for several other research activities, e.g., prototyping new radio schedulers, developing novel fine-grained orchestration algorithms involving CPU pinning or thread-level quota, or building on the GRC broker for the emulation of more complex scenarios with multiple users.

A. Scenario I: Dynamic orchestration

In this case, the objective is to stress the u-plane network functions. We start the experiment by deploying one mobile network instance, consisting of one 5GC, and one gNB with one associated UE, and configure the UE to runs an `iperf` test towards an `iperf` server connected to the UPF through the N6 interface. We measure the amount of CPU resources consumed by the different components via the Prometheous exporter, with the results being depicted in Figure 5. According to the results, the user plane function occupies most of the CPU, while the rest of the network functions have an almost negligible footprint.

To illustrate the dynamic orchestration capabilities of our system, which may be leveraged by e.g., AI algorithms in combination with the monitoring capabilities, we then trigger the instantiation of another network at approx. 1000 s. While this procedure is done manually for this experiment, in an autonomous environment the data shippers provide all the needed information about e.g., the network function load, to support this operation. As the figure illustrates, the two instances create a peak usage of 18% of the CPU, and again most of the resources are consumed by the UPF with minimal consumption by other network functions. According to these results, our platform can be used to emulate e.g. horizontal scaling scenarios, where additional resources are added to guarantee a given performance level.

B. Scenario II: UE Mobility

Next, we illustrate the ability of our platform to emulate situations where one or more UEs perform handovers between different cells. For this experiment, we rely on 4G connectivity since the srsRAN version we use (i.e., 22.04) does not support handovers on the 5G network.

We start by deploying a mobile network instance, with one EPC and two eNBs, where each eNB generates two cells. We then deploy two UEs, both associated with the same eNB but in a different cell, since the current implementation of ZeroMQ limits the number of UEs per cell to one. Like in the previous case, each UE generates bi-directional `iperf` traffic towards a server connected to the PGW-U.

From the above scenario, we trigger the handovers by taking advantage of the GRC broker to emulate a change in the channel conditions. More specifically, we program the GRC broker to emulate the movement of UEs between cells, by judiciously changing the received power from the UEs to each eNB (in the uplink) and from the eNBs to the UE (in the downlink). This is done by modifying the gains that multiply the samples from the eNBs, and throttling the samples from the UEs (see Fig. 3), achieving the effect of slowly moving away from one eNB and getting closer to the other one. The inverse procedure is applied to emulate the movement back to the previous cell, and the whole process is repeated indefinitely.

We depict in Fig. 6 the resulting downlink (top) and uplink (bottom) throughput at the MAC layer of the two eNBs. Firstly, it is worth noting the relatively small throughput values, which are caused by the emulation of the channel that "enlarges" the duration of the TTIs (thus decreasing the actual throughput). Secondly, the figure clearly illustrates how the throughput moves between eNBs, which approximately only one eNB sending and receiving traffic at a time. These results showcase the flexibility of the implementation of the GRC broker, which can be used and extended to support more complex radio scenarios, and be used as a sandbox for the development of mobility-triggered mechanisms (e.g., follow-the-load orchestration) in a controlled environment.

V. Related Work

The rapid, easy, and cost-efficient deployment of end-to-end mobile networks using open source software has been studied in various works and different implementations have been published mostly for large deployments. Works such as [11] and [12] leverage on the Open Air Interface (OAI) [13] as the RAN solution, using a hardware-based radio interface

(i.e., USRPs) as the front-end, thus preventing the evaluation of more complex channels conditions and mobility scenarios (note that channel emulation is available with the latest version of OAI). Additionally, the core network is provided and orchestrated as a single container without management capabilities at NF granularity and without providing control and user plane separation. In [14], certain extensions are proposed in Kubernetes resources' specification to support NFV loads. In general, the experimental work in this field focuses on the RAN component: for instance [15] leverages a network slicing aware modified version of srsRAN to provide a slice-aware orchestration solution. Still, as this setup is also leveraging real radio front-ends, the extensibility to more UEs or different radio conditions is limited. On the other hand, large-scale radio testbeds like Colosseum [10] have less focus on the core part, which is a fundamental aspect when testing end-to-end solutions. Hence, to the best of the knowledge of the authors, there has not been any open source effort for end-to-end mobile network stack emulation using cloud technologies, exclusively conventional hosts, and no RF hardware front-end, which would be ideal for small experimental deployments for research and educational purposes.

VI. Conclusion

Telecommunication operators' infrastructure is currently being migrated to cloud environments, with statically deployed PNFs being replaced by dynamic and flexible CNFs. The observability of heterogeneous network components is crucial to achieving predictability, fast fault detection, and scalability. With these new concepts being adopted, networking researchers need the ability to deploy end-to-end mobile networks for experimentation and development in an easy and affordable manner. In this paper, by leveraging existing open-source software projects and developing new modules and tools, we have designed and developed an end-to-end cloud-native open-source 4G/5G mobile network. This solution is a cost-efficient tool for the development and validation of novel control and orchestration algorithms since it does not require specialized hardware. We have illustrated the use of our solution with two use cases, leveraging the data shippers, the automatic deployment, and the channel emulation capabilities that have been developed.

Acknowledgements

This work is partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreements no.101017109 (DAEMON) and no. 101015956 (Hexa-X). This work is also partially supported by the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D 6G-CLARION and SORUS projects.

References

[1] D. Bega, M. Gramaglia, C. J. Bernardos Cano, A. Banchs, and X. Costa-Perez, "Towards the Network of the Future: From Enabling Technologies to 5G Concepts," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 8, p. e3205, 2017, e3205 ett.3205.

[2] A. Morris, "The Long Voyage to NFV: Vodafone And Orange Outline The Challenges Ahead — fiercewireless.com," https://www.fiercewireless.com/special-report/long-voyage-to-nfv-vodafone-and-orange-outline-challenges-ahead, [Accessed 25-Aug-2022].

[3] S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba, "Re-Architecting NFV Ecosystem with Microservices: State of the Art and Research Challenges," *IEEE Network*, vol. 33, no. 3, pp. 168–176, 2019.

[4] ETSI, "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Requirements for Service Interfaces and Object Model for OS Container Management and Orchestration Specification," Group Specification (GS) ETSI GS NFV-IFA 040, 11 2020, v4.1.1.

[5] "Cloud Native Computing Foundation landscape," https://landscape.cncf.io/, [Accessed 05-May-2022].

[6] K. C. Apostolakis, G. Margetis, C. Stephanidis *et al.*, "Cloud-Native 5G Infrastructure and Network Applications (NetApps) for Public Protection and Disaster Relief: The 5G-EPICENTRE Project," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC6G Summit)*, 2021, pp. 235–240.

[7] M. Gramaglia, P. Serrano, A. Banchs, G. Garcia-Aviles, A. Garcia-Saavedra, and R. Perez, "The Case for Serverless Mobile Networking," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 779–784.

[8] 3GPP, "Architecture Enhancements for Control and User Plane Separation of EPC Nodes," Technical Specification (TS) 23.214, 2016, v. 14.2.2.

[9] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "SrsLTE: An Open-Source Platform for LTE Evolution and Experimentation," in *Proceedings of ACM WiNTECH'16*. New York, NY, USA: Association for Computing Machinery, 2016, p. 25–32.

[10] L. Bonati, P. Johari, M. Polese *et al.*, "Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-the-Loop Network Emulation," in *Proc. of IEEE Intl. Symp. on Dynamic Spectrum Access Networks (DySPAN)*, Virtual Conference, December 2021.

[11] B. Dzogovic, V. T. Do, B. Feng, and T. van Do, "Building Virtualized 5G networks Using Open Source Software," in *2018 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*, 2018, pp. 360–366.

[12] O. Arouk and N. Nikaein, "5G Cloud-Native: Network Management & Automation," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–2.

[13] N. M. Rekoputra, R. Harwahyu, and R. F. Sari, "Performance Study of OpenAirInterface 5G System on the Cloud Platform Managed by Juju Orchestration," in *2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2019, pp. 211–216.

[14] B. Chun, J. Ha, S. Oh, H. Cho, and M. Jeong, "Kubernetes Enhancement for 5G NFV Infrastructure," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, 2019, pp. 1327–1329.

[15] G. Garcia-Aviles, C. Donato, M. Gramaglia, P. Serrano, and A. Banchs, "ACHO: A Framework for Flexible Re-orchestration of Virtual Network Functions," *Computer Networks*, vol. 180, p. 107382, 2020.

Nikolaos Apostolakis is a Ph.D. student at IMDEA Networks Institute, Madrid. He received his Diploma degree in Electrical and Computer Engineering at National Technical University of Athens and his Master degree in NFV/SDN for 5G Networks at Universidad Carlos III of Madrid.

Marco Gramaglia is a visiting professor at University Carlos III of Madrid, where he received M.Sc (2009) and Ph.D (2012) degrees in Telematics Engineering.

Pablo Serrano (M'09, SM'16) is an Associate Professor at the University Carlos III de Madrid. He has over 100 scientific papers in peer-reviewed international journals and conferences. He currently serves as Editor for IEEE Open Journal of the Communication Society.