Kube5G: A Cloud-Native 5G Service Platform

Osama Arouk and Navid Nikaein
Communications Systems Department - EURECOM, Biot, France - Email: firstname.lastname@eurecom.fr

***Abstract*—With the proliferation of use-cases envisioned to be supported by 5G networks, the focus is not only on the performance, but also on the service agility and elasticity. Cloud native principal is a methodology of designing lightweight, isolated-context, and deployable at scale applications that natively exploit the features of cloud. However, supporting telco applications (e.g., 4G/5G services) in the cloud brings up many challenges, such as the coexistence of physical and virtual functions, (near) realtime resource provisioning, service continuity, and strict latency and data rates. In this paper, we propose Kube5G, as a realizable agile service platform in support of 4G/5G cloud native applications. Kube5G introduces a novel approach in building and packaging a cloud-native compliant telco network function (NF) in a form of nested well-defined layers. In addition, we present a workflow for continuous integration and development operations in support of multi-version network functions (physical, virtual and cloud functions). We also present a concrete prototype implementation of Kube5G with experimental results indicating its efficiency and highlighting user and network perceived performance.**

***Index Terms*—5G, Cloud-Native, Mosaic5G, Kubernetes.**

I. Introduction

Fast, access at scale, frequent deployment of services, and quick failure recovery (e.g., service availability $\geq 99.999\%$) became important features to support current and novel 5G network services in divers virtualized/cloud environments. Supporting such flexibility in 5G networks as required to realize network slicing as well as service elasticity (e.g., service update/upgrade and scaling), can be achieved via softwarization, virtualization, and cloud computing technologies.

Cloud-native approach is a complete methodology to develop, build, run, and manage applications that fully exploit the cloud computing model. Cloud-native applications have three main design patterns: (1) *microservices*[1], where an app may be composed of many services that are meshed and operating independently from each other, (2) *containerization*, where an app is packaged in one or multiple isolated containers and managed by means of a set of standard APIs, and (3) *continuous integration and delivery (CI/CD)*, where an app goes through a fast cycles of development, build, test, release, and deployment. Twelve-factor[2] is a well-known approach for designing cloud-native software as a service (SaaS) including portability, isolation, stateless, unified and declarative APIs as a mean for inter-service communications. This paper leverages these principles in design and development of cloud-native telco applications, mainly in 4G/5G networking.

Microservices and containers are featured by small footprint, fast start time and comparative bare-metal performance, making them highly eligible for the deployment of 5G in the (edge) cloud. There are many containerization technologies, such as linux container (LXC), Containerd, CRI-O, and Docker, with relatively similar performance[3]. However, the number of containers grows drastically on per service basis (i.e., network slices and sub-slices), which calls for an efficient 5G service management and orchestration, able to achieve the envisioned performance, scalability, and agility. Several frameworks are already exist, such as Kubernetes, Docker Swarm, and Apache Mesos. For the ease of packaging, deploying and managing Kubernetes applications, Operator framework[4] is introduced under telcoKubernetes environment to encapsulate the lifecyle management operations and thus facilitate service automation. It has to be noted that telcoKubernetes is considered as defacto candidate for the orchestration of 5G and beyond in cloud environment as it has the largest community, better support, and promising future [1]. The same also holds for Docker container model[5].

Compared to IT applications, telco has many fundamental differences, making its adoption in the cloud a very challenging task. According to 5G-PPP [1], six fundamental telco features shall be supported in 5G. Here, we highlight the most important ones. Supporting meshing network service (as one application may be composed of many microservices) imposes the support of multiple networks per microservice. This requires extendable lifecycle management API to customize divers service operations. Telco applications and network functions (NFs) have very different behavior, some functions are light or non-real time while others are computational-intensive and/or real time, which may require particular hardware acceleration [1], [2]. Three types of networks functions are expected to co-exist: (i) physical network functions (PNFs) that either heavily rely on hardware (or implemented in a particular hardware) or are hardware assisted (i.e., running on specific hardware for e.g. acceleration), (ii) virtualized network function (VNF) that are software implemented and thus running on generic hardware, iii) cloud native network function (CNF) that are runnable against any platform. Supporting all of these network functions (NFs) types (i.e., multi-version functions) requires the awareness of the container management framework of the underlying platform's capabilities that needs to be supported. Different from the classical cloud applications, telco applications are geographically distributed across multiple datacenters. Therefore, multi-site support is of paramount for 5G.

Previous works, as elaborated in the next section, represent extreme points in the design space and therefore have managed to only partially address some of the aforementioned

[1] https://martinfowler.com/articles/microservices.html
[2] https://12factor.net/
[3] https://kubedex.com/kubernetes-container-runtimes/
[4] https://coreos.com/blog/introducing-operators.html
[5] https://bit.ly/3ekwpvT

objectives. In this paper, we propose a cloud native framework that is tailored to telco applications. The design of Kube5G natively supports a multi-version 5G CNF, in particular during the CI/CD process. The lifecycle management in Kube5G is automated and supports advanced operations such runtime reconfiguration, and failover. In summary, this paper makes the following contributions:

- (Section III) a realizable cloud native 5G service platform in the form of Kube5G;
- (Sections IV) a novel framework for building and automating the lifecycle of 5G cloud-native NFs, as well as concrete implementation details of Kube5G based on OpenAirInterface (OAI), Mosaic5G, Ubuntu Snaps, Docker and Kubernetes;
- (Section V) experimental evaluation of the various aspects of Kube5G highlighting its performance as well as various capabilities compared to the state-of-the-art solutions.

II. Related Work

In the literature, there are few works about the automation of the deployment of containerized 5G NFs. The authors in [3] propose a cloud native solution for scaling the core network (CN) entity MME (Mobility Management Entity) for handling the control signaling overhead from Radio Access Network (RAN). The authors in [4] propose a 5G-aware testbed, with the support of containers and virtual machines orchestration. Another platform in the vein of 5G is the work in [5], where the authors propose a platform as a service for the development and operation of network services. The authors claimed that the platform offers better flexibility and also lowers the barriers for unifying the broad spectrum between technologies and services in telco landscape. However, these later works do not focus on how to build and automate the lifecycle of 5G functions in a cloud environment. The focus of the authors in [6] is more about how to build a 5G mobile network using the open-source OpenAirInterface (OAI), without any focus on how to build the 5G functions and automate them in a cloudified environment. With a focus on the RAN as open-source for Mobile Edge Computing (MEC) applications in the context of 5G networks, the authors in [7] present an automated way using devops for the development and deployment of 5G MECs. However, none of these works tackle the problem of building and automating the lifecycle of 5G NFs following cloud-native computing principles. In this paper, we present a novel framework for *designing and building* 4G/5G cloud-native NFs and *automating* the lifecyle of 4G/5G network services for both RAN and Core Network (CN) domains. It is worth noting that there are many opensource projects fostering the deployment of 5G and beyond in cloud native environment, such Open-RAN (O-RAN), and Cloud Native Computing Foundation (CNCF)[6]. Further, many efforts have been made to build operational 5G testbed facilities in Europe, and US considering cloud native design patterns, namely 5G-EVE, 5G-VINI, 5G-GENESIS, EMPOWER (EU-US) and PAWR.

[6] O-RAN: https://www.o-ran.org, CNCF: https://www.cncf.io

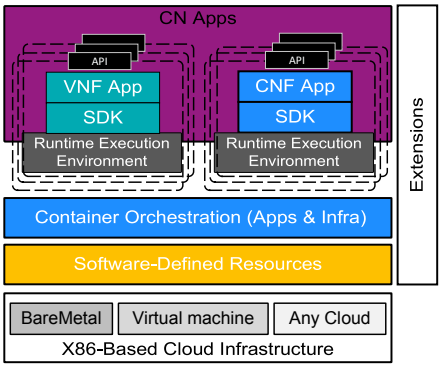

Fig. 1: High-level Architecture of Kube5G

III. Kube5G overview

The core contribution of this paper, Kube5G, is a novel cloud native 5G design that is inline with the spirit of CNCF and the needs of an agile, scalable, and on-demand service-oriented mobile network architecture. Being able to simultaneously satisfy (a) building and packaging cloud-native 5G NFs, and (b) runtime service automation and capability exposure in the context of 4G/5G RAN and CN, is the main, yet not resolved, challenge addressed by Kube5G. Kube5G architecture is illustrated in Fig. 1 that is composed of (from bottom to top) (i) *infrastructure and Software defined resources*, (ii) *container orchestration*, (iii) *runtime execution environment*, (iv) *packaged cloud native applications*, and (v) *extensions*. The *architecture* (ideally) supports any type of deployment, regardless whether it is baremetal, virtual machine (e.g., virtualbox), or any private/public/hybrid cloud. In order to fully optimize the resources, future data centers shall support full disaggregation and abstraction of resources [8]. The *container orchestration* acts as both infrastructure manager, i.e. cloud-native infrastructure manager (CIM), and application manager, i.e. cloud-native network function manager (CNFM) [9]. Thus the *container orchestration* is responsible of controlling infrastructure resources like compute and networking. Like virtualized infrastructure manager [9], CIM has many responsibilities/functionalities, such as orchestrating the allocation/upgrade/release/reclamation of resources (including resources' optimization), and managing the mapping of virtual-physical resources. As CNFM, *container orchestration* is responsible of i) CNF instantiating and (pre)configuration, ii) CNFM software update and/or upgrade/downgrad (whenever needed) and CNF scaling, iii) CNF automated healing and termination and lifecycle management change notification and monitoring [9]. Kube5G supports the co-existance of both VNF and CNF as currently required for 4G/5G NFs. VNF is any virtual function not developed to be cloud native compliant, which are generally monolithic functions or environments and/or hardware-dependent NFs or entities. In order to allow VNFs to be executed in a cloud native environment (e.g., Kubernetes), the runtime execution environment is needed, such as Virtlet and Kubevirt. As detailed later in the design of a cloud-native application, supporting APIs as binding ports is important for CNFs for communications between services, as

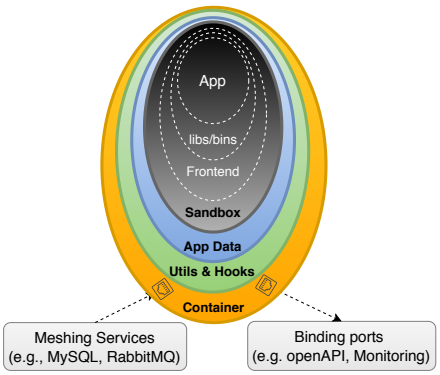

Fig. 2: Nested-layer design of cloud native-compliant NF

well as remote interactions with them. Further, packaging the application with software development kit (SDK) and APIs in containers provides agility, isolation and portability, and thus the operability against any platform. In this context, SDK is a set of layers decoupling an application from its data, tools, and hooks, and provide specific functions and methods to be accessed through one or more API calls (Fig. 2).

Finally, the *extensions* play an important role in Kube5G, in order to support many value added functionalities. Examples of such extensions are: (i) management and orchestration (MANO) like OSM, (ii) monitoring and network intelligence like kubeflow[7], (iii) cloud native network deployment and automation via Kube5G-Operator[8] that is openshift operator.

IV. Kube5G Design and Implementation

A. cloud native-compliant network function (NF) design

Fig. 2 illustrates the proposed novel design of a cloud-native NF as a set of nested layers, compliant with the twelve factor methodology. Layers are loosely coupled with each other to allow independent lifecycle management process to occur. Similar in spirit with Docker, the nested-layer design allows reusability across layers. This is of paramount importance in cloud environment to (a) minimize application update/upgrade time due to CI/CD, (b) retain service continuity when updating/upgrading the application by keeping application data outside of the sandbox, and (c) allow application portability across different execution environments. In the following, we present each layer separately.

Sandbox*:* isolates the application from its runtime execution environment and grant access to resources required for proper operation of the app (e.g., analogy with Android and IOS apps). It is a self-contained package, where all application's dependencies are embedded, thus achieving application portability. Snap[9] is a good example of sandbox. As illustrated in Fig. 2, the sandbox contains i) the application, ii) its library and binary dependencies, iii) frontend as a value added services such as versioning, capabilities, monitoring and configuration APIs, exposed by the application.

[7] https://www.kubeflow.org/

[8] http://mosaic-5g.io/kube5g/

[9] https://snapcraft.io/docs/getting-started

App Data*:* stores all the application-specific data required for service elasticity (e.g., redeployment, scalability). It includes credentials, configuration, and runtime stats and logs stored across different app revisions. Application frontend is in charge of maintaining **App Data**. Such a decoupling makes the application stateless and thus significantly improves the efficiency of lifecycle management processes.

Utils and Hooks*:* **Utils** are a set of helper tools that are needed inside the container to perform runtime testing or monitoring of the application. They are used to realize service scalability and agility (e.g., monitoring the load and scaling up/down the resources). On the other hand, **Hooks** are in charge of application lifecycle management inside the container, such as application restarting, (re)configuration, or sandbox upgrading.

Container*:* is a software package containing everything required to run the application properly. Different from **Sandbox**, **Container** includes system tools, libraries, and settings, thus making the software completely independent of the system on which the application is running. One of the main important features of **Container** is its elasticity, where many instances of the same application can be run on the same host, or being distributed over many environments. **Container** supports two types of ports: (1) binding ports needed to connect to external applications as service producer/consumer for the purpose of logging and monitoring (e.g., flask and OpenAPI), and (2) meshing service needed to connect to external applications and share states for a proper operation of the embedded application (e.g., database). This port is also used to store the **App Data** outside of the container when needed (e.g., container upgrade or failover).

B. CI/CD Workflow

Fig. 3 illustrates the proposed workflow of continuous integration and deployment in support of 4G/5G NFs. All the different versions of NFs (i.e., PNF, VNF, and CNF) are supported in this workflow. The workflow starts when there is a change/update in the project (application), where these changes are pushed to the version control (e.g., git/gitlab) that will trigger an automatic testing of the change/update (e.g., jenkins), which in general includes run/build/test the code. After successfully testing the code and if the application exists in a form of PNF (i.e., it needs to run on specific hardware like accelerator), the application is pushed to PNF registry. After that, this PNF function will be registered in the PNF catalog of management framework (MF). MF is the orchestration framework that manages and orchestrates the deployment of multi-version functions in the underlying cloud infrastructure [1]. Note that for the other type of PNF function, i.e. functions implemented as hardware, it is assumed that they are already registered in the PNF catalog when the underlying platform exposes its capabilities (Fig. 3). The next step is to create the sandbox version (refer to Fig. 2) of VNF by passing the NF to sandboxing operation. Similarly, the sandbox version of the application will be registered in sandbox registry (e.g., pushing the new version of sandboxed app to snapcraft store), and MF will register this function in the VNF catalog. The

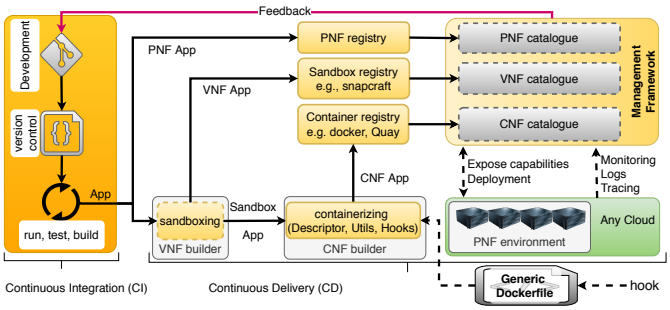

Fig. 3: Kube5G CI/CD workflow

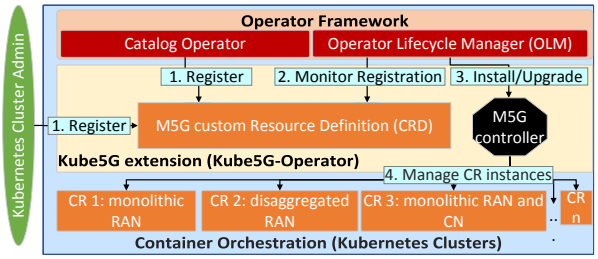

Fig. 4: Kube5G-Operator workflow

final step is to containerize the application (refer to Fig. 2) by passing its sandbox version into containerizing operation, and then push it to the container registry. The containerization is done using one of the containerization technologies such as Docker or Mesos. Finally, this function will be registered in the CNF catalog of MF.

C. CNF implementation

We realize 4G/5G CNF implementation by leveraging Ubuntu Snap as a sandboxing technology, Linux shared memory to store app data, docker hooks and python libs to manage the application, and Docker to containerize the Mosaic5G NFs. Since Mosaic5G functions[10] are sandboxed in snap version, we exploit them to build the Docker containers. Notice that the OAI source code used to build the Mosaic5G snaps is developed following Devops principles[11]. Docker provides two ways to run an application: (i) get the application's Docker image from Docker's store (e.g., Docker hub), or (ii) build the Docker image of the application using Dockerfile. Dockerfile can be defined as a script file containing all the commands that a user needs to call on the commandline for doing everything to make the application runnable inside the Docker container. In order to automate the creation of Docker images using the same Dockerfile, we create build script (c.f. CNF builder in Fig. 3). The configuration and running of Mosaic5G snaps are done using hooks, which are copied inside the Docker image during build process. Note that the hooks act as an application interface (e.g., entrypoint) inside the container.

D. Operator service automation

In order to automate the application lifecycle management in runtime, we use Kubernetes and develop a Kube5G-Operator[12] on the top of Operator framework. Kubernetes introduces a powerful tool, called Custom Resource Definition (CRD), allowing users to create new types of resources. With CRD, Kube5G-Operator exposes 4G/5G network service as a single object that only exposes service-specific APIs that make sense for service providers rather than a collection of primitives like Pods, Deployments, Services or ConfigMaps. To this end, Kube5G-Operator implements and automates both basic (e.g. installation, configuration, and monitoring) and advanced (e.g. re-configuration, update, backup, failover, restore) operations in a form of a portable Kubernetes-native applications running inside a Kubernetes cluster. In addition, it is capable of managing complex 5G services (e.g., flexible network deployment), which is traditionally managed by administrators and their operational scripts (e.g., Ansible).

As shown in Fig. 4, the Operator utilizes what is called Controller. Multiple controllers may exist, depending on the capabilities of the Operator, to monitor a certain CRD. Considered as one of the core concepts of Kubernetes, a Controller is an entity that continuously loops on the master node of Kubernetes, while listening to the cluster for detecting any changes. If an event happens on a watched resource type like failure of a pod, a reconcile cycle will start. This is done through the reconciler object associated with the concerned controller, where every controller has a reconciler object. The output of the reconcile function will decide whether it should be run again or the reconciliation is finished. For example, a controller may listen to any changes to its pods, and makes changes if the state of pods is incorrect. On the top of the Operator lies Operator Lifecycle Manager (OLM) for managing openshift Operators like installation and update, as well as managing their associated services running across Kubernetes clusters. In order to make such automation more dynamic, we expose a set of configuration parameters to the users that facilitates runtime service reconfiguration, e.g. DNS, bandwidth, and frequency bands[13]. The capabilities currently supported by the Kube5G-Operator are: (i) deployment and release, (ii) configuration and reconfiguration, (iii) network update, upgrade, and downgrade, (iv) network re-aggregation, disaggregation, and (v) CNF monitoring (health and metrics). It has to be mentioned that the current implementation of Kube5G-Operator does not fully support the insight and autopilot features of an Operator[14].

V. Performance Evaluation

In this section, we assess experimental results related to the efficiency of Kube5G platform as well as the perceived user and network performance. We used OAI and Mosaic5G platforms to deploy an operational 4G+ network that supports disaggregated RAN (i.e., flexible functional split, and in particular F1 interface). In our setup, a private cloud consisting of two powerful workstations under Ubuntu 18.04, is used. The remote radio head is built based on the USRP B210, duplexer, and antenna operating in FDD SISO mode, Band

[10] http://mosaic-5g.io/store/
[11] https://bit.ly/3c5RyYW
[12] https://gitlab.eurecom.fr/mosaic5g/kube5g/tree/master/openshift/m5g-operator
[13] https://hub.docker.com/r/mosaic5gecosys/oairan
[14] https://www.openshift.com/learn/topics/operators

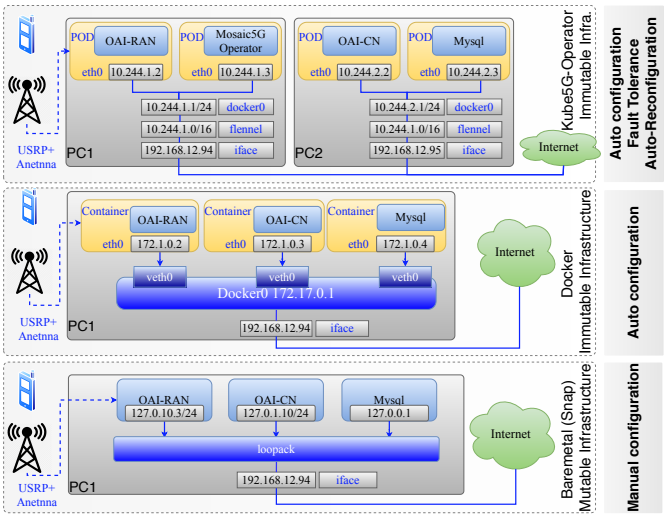

Fig. 5: Network topology setup

7, with 5 MHz channel bandwitdh. The user equipment (UE) is an iPhone SE.

A. Performance Metrics

We compare and analyze the lifecycle operations in a setup with OAI Snaps[15] in three settings: (i) bare metal (Snap), (ii) Docker and docker-compose, and (iii) Kube5G-Operator. Note that Snap deployment is considered as mutable infrastructure, while Docker and Kubernetes are immutable infrastructure [10]. The three considered network setups are shown in Fig. 5, where two evaluation metrics have been considered:

Deployment and reconfiguration time: The deployment time represents both the download and provisioning time. The former is the time to download the snaps for the bare-metal case or to pull the Docker images of OAIs from Docker hub for the two other cases[16], while the later is the time to bring up the network service til the users start consuming the service. Thus the provisioning time includes the configuration time plus the deployment time. Concerning the case of snaps, it is assumed that the provision time of the snaps is done semi-automatically using different scripts that are run after the download. As for the memory footprints, sandboxed snaps and their associated Docker images are designed and built to support different deployment scenarios ranging from monolithic to disaggregated. For example, OAI-CN snap and its Docker image may act either as a standalone core network or just as one network entity such as AMF/MME (Access and Mobility Management Function/Mobility Management Entity).

Network and user experience: we consider two different measurements: (i) TCP/UDP performance using iperf3 , (ii) application level performance using nperf[17]. The nperf tool includes: DL/UL rates, latency, browsing and stream performance percentage index. Note that the browse (navigation) performance reflects the loading time of different websites, while the video performance index is the ratio of loading and playing time of video to its nominal duration.

[15] https://snapcraft.io/search?q=mosaic-5g

[16] https://hub.docker.com/u/mosaic5gecosys

[17] iperf3:https://iperf.fr/, nperf:https://www.nperf.com

B. Results

1) Deployment and reconfiguration time: Fig. 6 shows the download, service provision, and runtime reconfiguration time, for the considered deployments. It is clear that the download time for Docker and Kubernetes is more than the double compared to that of snaps. This is because of a larger memory footprint of Docker images compared to Snaps, i.e. 528MB for CN and 745MB for RAN images versus 58MB for CN and 247MB for RAN Snaps. Although it takes about 30 seconds more to download the images, the benefit of using containers is clear when comparing with the service provision time (configuration and deployment time until connecting the UE). It can be observed from the figure that the service provisioning time is higher by a factor of 3 for snaps compared to Docker or Kube5G-Operator. Concerning Kube5G-Operator, the mean time for provisioning a service is about 1 minutes and 12 seconds, which takes (on average) 16 seconds more compared to docker. Therefore, service automation can be significantly improved using docker and Kube5G-Operator framework.

When comparing service runtime reconfiguration, it can be seen from Fig. 6 that the Kube5G-Operator is able to lower the time by a factor of 2 compared to Docker and baremetal (Snap). The main reason behind this is that the Kube5G-Operator redeploys only those entities for which a custom resources (CR) is applied (e.g., reconfiguration) while retaining the others[18], whereas Docker redeploys the whole network service. Notice that two reconfiguration times are measured for Kube5G-Operator: (i) both RAN and CN reconfiguration, and (ii) only RAN reconfiguration. In the former case, the reduction in reconfiguration time for Kube5G-Operator comes from the definition of CR that reconfigures RAN and CN without changing the HSS database (mysql in Fig. 5). It can also be observed that the Baremetal (Snaps) reconf has a lower reconfiguration time than that of Docker and very comparable time with respect to Kube5G-Operator. This is because both Docker and Kubernetes-Operator are based on an immutable infrastructure [10], and thus require to redeploy the network (fully and partially), whereas Snaps just require (manual) service reconfiguration.

It has to be noted that when a failure happens, network-wide redeployment process may be triggered for both Docker and Baremetal (Snap). Instead, Kube5G-Operator leverages CRD to recover from a failure, for example by redeploying those entities concerned by the failure[19] or network auto-configuration. Moreover, it is even very complex and not feasible to reconfigure many tens (even hundreds) of entities without breaking/degrading the service when using Baremetal (Snap) or Docker. The above results demonstrate that Kube5G-Operator significantly improves the efficiency of service automation in terms of lifecycle management, scalability, and failover when compared to Baremetal (Snap) or Docker.

2) User and Network experiences: Fig. 7 illustrates the sustainable TCP and UDP performances. From the figure, it is

[18] The deployment is done without scaling, i.e, there is one instance of an entity

[19] https://bit.ly/36EeEEV, https://bit.ly/2X7jYO4

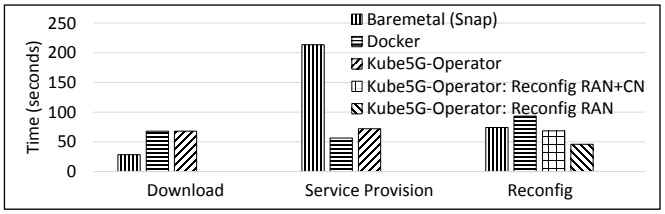

Fig. 6: Download, Service provision, and reconfiguration times

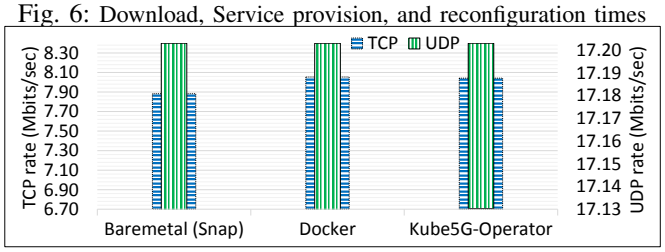

Fig. 7: TCP/UDP traffic for the considered deployments

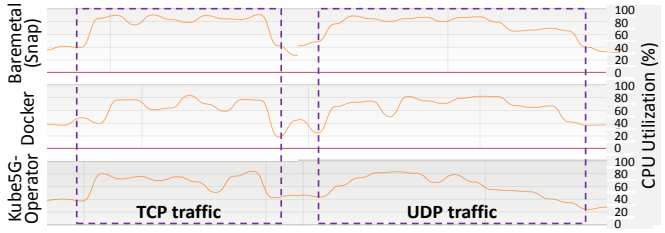

Fig. 8: Instantaneous CPU utilization for TCP/UDP traffic

clear that the deployments (namely baremetal (Snap), Docker and Kube5G-Operator) have comparative performance. The difference in the mean TCP traffic between baremetal (from one side) and Docker and telcoKubernetes (from the other side) is about 1%, while it is close to zero between Docker and telcoKubernetes. Regarding UDP traffic, the difference in the mean value between different deployments is close to zero. Instantaneous CPU utilization when generating TCP and UDP traffics is shown in Fig. 8, where only one CPU is exploited to carry the traffic. It can be observed that the CPU utilization for TCP traffic is more fluctuating for Snap and Docker than that of Kube5G-Operator, mostly due to bidirectional behavior of TCP. The maximum CPU utilization in case for Snap is 90% (both TCP and UDP), and for Docker and Kube5G-Operator is 80% (UDP) and 85% (TCP). These results reveal also that the CPU resource utilization profile for Kube5G-Operator is more efficient and stable than that of Snap and Docker.

Fig. 9 shows application level performance for the three considered deployments. Again, it can be observed that they achieve a comparative performance, with a slightly better score for Kube5G-Operator. These results confirm the feasibility of cloud native deployments of 4G/5G network service while retaining user and service perceived performance. This indicates that the 4G/5G network, and more generally telco applications, can be provided on as-a-service basis in a very short time scale exploiting service elasticity, scalablity, and agility.

VI. Conclusions

Cloud-native technologies are the main enabler for efficient service deployment, fast recovery, agility, elasticity, and access at scale for 5G services. In this paper, we present Kube5G as an agile service platform for supporting 4G/5G networks in

(a) Baremetal (Snap) (b) Docker (c) Kube5G-Operator

Fig. 9: Network and User experience

cloud-native environments. Kube5G proposes a novel CNF design as a nested reusable layers to ease building, packaging, and upgrading of multi-version NFs during CI/CD process. A concrete implementation of 4G/5G cloud-native functions is presented with automated lifecycle management operations using Kube5G Operator. Kube5G is benchmarked against Baremetal (Snap) and Docker deployments demonstrating its feasibility and efficiency for fast roll-out of 4G/5G network services. The results confirm that 4G/5G network can be provisioned in less than two minutes, and updated/reconfigured (as a part of auto-pilot feature) in less than a minute. In future, we plan to model and support natively near-realtime CNF for 5G applications and services in Kube5G.

Acknowledgment

The research leading to these results has received funding from Davidson Consulting as well as European H2020 5G-Victori and Affordable5G projects under grand agreement 857201 and 957317.

References

[1] 5G-PPP, "From Webscale to Telco, the Cloud Native Journey," 5G-PPP Software Network Working Group, Tech. Rep., 08 2018.

[2] N. Nikaein, E. Schiller *et al.*, *Towards a Cloud-Native Radio Access Network*. Cham: Springer International Publishing, 2017, pp. 171–202.

[3] P. C. Amogh, G. Veeramachaneni *et al.*, "A cloud native solution for dynamic auto scaling of mme in lte," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017, pp. 1–7.

[4] L. T. Bolivar, C. Tselios *et al.*, "On the deployment of an open-source, 5g-aware evaluation testbed," in *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, March 2018, pp. 51–58.

[5] S. Van Rossem, B. Sayadi *et al.*, "A vision for the next generation platform-as-a-service," in *2018 IEEE 5G World Forum (5GWF)*, July 2018, pp. 14–19.

[6] B. Dzogovic, V. T. Do *et al.*, "Building virtualized 5g networks using open source software," in *IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*, April 2018, pp. 360–366.

[7] J. Haavisto, M. Arif *et al.*, "Open-source rans in practice: an over-the-air deployment for 5g MEC," *CoRR*, vol. abs/1905.03883, 2019.

[8] S. Han, N. Egi *et al.*, "Network support for resource disaggregation in next-generation datacenters," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013.

[9] ETSI GS NFV-MAN 001 V1.1.1, "Network Functions Virtualisation (NFV); Management and Orchestration ," December 2014.

[10] K. Hightower, B. Burns, and J. Beda, *Kubernetes: up and running: dive into the future of infrastructure*. " O'Reilly Media, Inc.", 2017.