Automated orchestration of virtualized deployments of mobile core networks.

Piotr Borylo[a,*], Wojciech Stasiak[a], Piotr Piwowar[a], Nashid Shahriar[b]

[a] *AGH University of Krakow, Institute of Telecomunications, al. Mickiewicza 30, Krakow, 30-059, Poland*
[b] *University of Regina, 3737 Wascana Parkway, Regina, S4S 0A2, Canada*

Abstract

Given the ongoing process of softwarization and virtualization of communications networks, including 5G and beyond 5G (B5G), automation has become an inevitable part of network deployment. Ensuring a robust, efficient, and zero-touch deployment process is expected to drive the advancement of B5G and 6G technologies. Automation will become increasingly critical because of increasing complexity and requirements on dynamic provisioning of these networks. A well-designed automation process can streamline the deployment of private mobile networks for Industry 4.0 or on-demand networks for mass events, ensuring efficient video signal transmission and reducing the need for physical wiring. This paper presents a Proof-of-Concept testbed tailored for the automated orchestration of virtualized deployment of a 5G mobile core network, along with insights into potential challenges and solutions. Furthermore, we compare two open-source core network implementations in terms of feasibility for virtualization, automation and performance of user plane (delay, throughput) and control plane (user equipment registration time and PDU session establishment time). We evaluate the performance of automated deployment of 5G core network locally in the testbed and based on the results we propose enhancements that reduce deployment time by 40% on average for one of the implementations. We also present the process and necessary tools to automatically orchestrate deployment of a virtualized 5G core network in a cloud infrastructure. Finally, we performed preliminary performance assessments of the automation process in the proposed cloud-

*Corresponding author
Email address: piotr.borylo@agh.edu.pl (Piotr Borylo)

based deployment.

Keywords: network softwarization, network deployment automation, 5G core network, control plane performance, Proof-of-Concept testbed
2000 MSC: 9405, 94A99

---

1. Introduction

The emergence of 5th Generation (5G) mobile network, as a successor to LTE, offers higher data rates, reduced latency, and improved connection density [1]. These advancements are made possible by leveraging Network Function Virtualization (NFV), which allows for the virtualization of network functions, separating them from underlying hardware and enabling software-based implementations. This virtualized infrastructure presents an ideal opportunity for automation, as it provides flexibility, scalability, and eliminates the need for costly hardware upgrades.

Automated orchestration of mobile core networks is further justified by features like network slicing, which divides the network into isolated instances tailored to specific challenges such as application support and Quality of Service (QoS) parameters [1]. Automation is crucial for efficiently provisioning and maintaining a vast number of slices, especially due to the challenges imposed by the increased number of services and complexity of future technologies like B5G and 6G [2]. Furthermore, components of the 5G network can be deployed either in a centralized (monolithic) architecture or distributed across several physical or virtual machines. Thus, automation can facilitate the challenging placement of network functions at various locations, including the cloud infrastructure and network edge with the use of Multi-access Edge Computing (MEC), allowing for customized and low-latency services [3].

Automated cloud-based deployments of virtualized core networks is one of the most emerging challenges. To the best of our knowledge, it has not been properly addressed yet. By embracing automated service orchestration, virtualized mobile core networks can address the challenges of dynamic infrastructure provisioning, as outlined below. Automation can be especially beneficial in case of distributed deployments, which will be critical for B5G and 6G deployments.

First, automation addresses challenges related to manual configuration and intervention, minimizing human errors and streamlining the deployment process. Second, automation is essential for managing the complexity of 5G

networks by orchestrating and coordinating various elements of the deployment process. Moreover, the dynamic nature of B5G and 6G networks, with their varying traffic patterns and demands, necessitates adaptable deployments, adding further complexity and challenges. Additionally, the rapid evolution of technology and the frequent introduction of new services and features in the 5G ecosystem require quick and agile deployments to meet these demands. Lastly, automating 5G network deployment helps address business challenges by minimizing operational expenses and capital expenditures through reduced manual labor, streamlined processes, and optimized resource utilization.

As we move towards an era of on-demand deployment and increasingly challenging use cases like vehicle to anything (V2X) communication, Industry 4.0, temporary mobile networks for mass events, and Information Centric Networking [4], automated orchestration of virtualized mobile core networks becomes an inevitable and essential component. That is why, in this paper, we address challenges of dynamic and robust provisioning of the 5G core network. Specifically, we develop a comprehensive Proof-of-Concept testbed of automated virtualized deployment of such network and make corresponding automation scripts publicly available [5]. We provide valuable insights on building similar infrastructures and additional hints about potential problems and solutions for them. We also propose a solution to automatically deploy a virtualized 5G core network in a cloud infrastructure (e.g., OVH Cloud[1]).

The contribution of our paper can be summarized as follows:

- an automation framework tailored for 5G core network deployment comprising technology stack, configuration files and comprehensive descriptions facilitating reproducibility of the environment and removing the initial impediments in the 5G deployment process,

- performance evaluation of user plane (delay, throughput) and control plane (user equipment registration time and Protocol Data Unit session establishment time) of two open-source core network implementations (Free5GC [2] and Open5GS [3]),

[1]https://www.ovhcloud.com/
[2]https://www.free5gc.org/
[3]https://open5gs.org/

- performance evaluation of the automated 5G core network orchestration (e.g. time needed to deploy core network) and discussion on the feasibility for virtualization and automation of aforementioned core network implementations,
- modifications to the existing tools removing the initial impediments in the automated 5G deployment process due to the required human intervention and facilitating 5G core network deployment,
- original enhancements that reduce deployment time by 40% on average for one of the tools (Open5GS),
- the process and necessary tools to automatically deploy a distributed virtualized 5G core network in a cloud infrastructure and some preliminary evaluation results (for this purpose we selected a more prominent open-source core network implementation).

Our contribution enables rapid deployment of 5G mobile core networks while facilitating analysis of critical aspects and informing architectural decisions regarding tools and technology stack. Thus, our paper facilitates industry and academia while experimenting and developing automated orchestration systems for mobile core networks. Our contribution is particularly significant given the increasingly challenging requirements imposed by B5G and 6G networks on these orchestrated deployments. Meeting these requirements will be obligatory to provide emerging services and satisfy quality of service requirements. The presented results and discussion inform architectural decisions and suggest future research directions.

In this work, we propose the automation framework for the deployment of Standalone (SA) 5G core networks. The framework includes, among others, technology stack, configuration files, images of virtual machines, modifications to the virtualization environment, and configuration of kubernetes cluster. All components are specifically designed for 5G networks, making our solution tailored for this application. By developing a framework dedicated to the 5G core network, we were able to address challenges overlooked in the existing state of the art. For instance, we focused on ensuring seamless networking between the 5G architecture components distributed across cloud locations, while incorporating specialized protocols. Additionally, the automation process includes a comprehensive configuration of 5G network elements.

The framework is further utilized to conduct an extensive experimental-based performance assessment of the automation process, providing a comparison of different implementations of the 5G core network. In the subsequent section, we provide an overview of existing works in the related area and discuss how our paper contributes to the research field. Section 3 introduces the fundamentals of 5G networks and highlights the challenges associated with automated deployment. In Section 4, we present our main contribution, which is the automation framework for 5G mobile networks. The section provides detailed insights into the design and implementation of the framework. Furthermore, we present the experimental results obtained and provide a comprehensive analysis in Section 5. Section 6 describes the proposed deployment process and the tools we used to automatically deploy a virtualized 5G core network in a cloud infrastructure. Based on the discussion and results presented in Sections 3 and 5, respectively, we selected Open5GS as a more prominent candidate for a cloud-based deployment. Section 6 also contains preliminary performance assessments of the automation process in the proposed cloud-based deployment. Finally, Section 7 concludes the paper, summarizing our findings and discussing potential avenues for future research.

2. Related works

This section provides an overview of related works based on the following taxonomy. First, we discuss papers that specifically concentrate on evaluating the performance of 5G core network deployments and comparing different implementations. Subsequently, we analyze works that focus primarily on the automation aspects of 5G deployments. Finally, we explain the relationship between our work and orchestrators of services and network functions.

Rischke et al. [6] conducted a measurement study on a campus 5G network to compare the performance of Stand Alone (SA) and Non-Stand Alone (NSA) 5G network deployments. The experiments involved Nokia radio equipment and evaluated the performance of Nokia 5G core network in the NSA scenario, as well as the Open5GS core network in the SA scenario. The study focused on measuring and analyzing delays and losses in the network. Similarly, authors of [7] introduced an innovative testbed that incorporates both 4G and 5G mobile networks, leveraging open-source solutions. Deployments were compared with the respect to throughput, latency and received signal strength. Achieved results were compared with commercial network.

While both papers offer valuable insights into the performance characteristics of the network, it does not provide a direct comparison between different SA 5G mobile core network implementations and does not address the automation issues. The performance issues are also considered for radio part of 5G networks in [8]. The authors in [8] measure transmission parameters in an interesting testbed of SA indoor 5G network. They utilize commodity hardware and open source implementation of 5G control plane to measure latency, data rate and coverage. Complementary, authors in [9] performed a CPU profiling for software components of a particular implementation of 5G radio access network. The primary objective of this study was to identify the resource-intensive software components within the distributed 5G RAN architecture, enabling more efficient code optimization and hardware offloading techniques. However, both [8] and [9] focus mostly on the RAN, while mobile core networks and automation issues are neglected.

Neto et al. [10] conducted an assessment of open-source implementations of the core network, specifically focusing on Open5GS, Free5GC, and Magma Core projects. The paper provides a theoretical evaluation of these implementations, primarily discussing their functional aspects. The evaluation encompasses various factors such as architecture, documentation, community support, and management. A tutorial about selected communication protocols used in the 5G networks is the main part of the work [11]. The comprehensive analysis covers Non-Acces Stratum (NAS) and Next Generation Application Protocol (NGAP) concerning the protocol stacks and functionalities related to session management and resource allocation. In the theoretical part, the authors reviewed communication sequences for NAS and NGAP. In the practical part, the authors applied a black-box testing approach with the use of an original 5G tester. They evaluated the conformance and robustness of three open-source 5GC projects. The aim was to verify the quality of the control plane implementations. Thus, both works should be considered complementary to our study, which is focused mostly on the comparison of performance parameters of different 5G core network implementations. Furthermore, none of these papers specifically address the automation aspects of 5G networks.

The works [12] and [13] primarily focus on experimental performance assessment. Galibondo et al. [12] specifically present the deployment of a 5G private network using open-source solutions, encompassing both radio and core network components (including Open5GS, which is further explored in this paper). The authors validate the interoperability of the composed sys-

tems by combining different RANs with different core networks. Multiple tests were conducted, including measurements of latency and packet loss, with all cases showing a packet loss rate of 0%. The authors also examine various implementations of the core network. A recent work presented in [13] primarily focuses on comparing the performance of two open-source implementations of the 5G core network, namely Free5GC and Open5GS. While the paper introduces an original test tool, our study utilizes a well-known and popular simulator for the 5G access network. The conclusions drawn in [13] suggest that Free5GC exhibits better user plane performance, whereas Open5GS is regarded as a more stable solution with a more reliable registration process. Although our work and [13] employ similar metrics such as UE registration time, user plane throughput (we additionally measured latency), and resource usage (RAM in our case and CPU in [13]), both studies complement each other due to their exploration of core networks deployed using different techniques. Moreover, while [13] and the aforementioned works primarily focus on deployment performance, our paper has a greater emphasis on the automation of 5G core network deployment and its robustness. Namely, we propose the framework to automatically deploy a distributed virtualized 5G core network in a cloud infrastructure, evaluate the automation performance, and propose a solution to enhance the automation process.

Lake et al. [14] conducted a comprehensive survey focusing on the softwarization and programmability, which are key drivers for automation in this field. The authors provide a detailed overview of the existing technologies and tools that facilitate the deployment of 5G networks, encompassing both the core and radio aspects. Notably, they highlight two core implementations, namely Free5GC and Open5GS, which serve as prominent platforms in this domain. One crucial observation made by the authors is the need to strike a balance between the efficiency and flexibility of 5G deployments in order to achieve interoperable solutions. This trade-off underscores the complexity inherent in designing and implementing 5G networks. The survey further showcases a wide variety of tools that can be potentially utilized in this context, from which we narrow down and select components for the framework in our work.

Wiranata et al. [15] propose an approach to automate a 5G network utilizing Kubernetes and Mosaic 5G Operator. Their framework comprises both the radio and core components of the network. Notably, the authors employ a 4G core network in conjunction with 5G wireless equipment. The primary focus of their study lies in the orchestration of the deployment, with

particular emphasis on evaluating the throughput. The architectural design presented in the paper incorporates monitoring functionalities facilitated by ElasticSearch and Kibana. However, it is important to note that while this work offers solutions for automation of 4G network, it lacks a comprehensive analysis of the performance of the automation process itself. Additionally, the paper does not provide any comparative assessment of available SA 5G core network implementations. Although the presented results are important and meaningful, they tend to be more qualitative and selective rather than quantitative. Similarly, a github repository [16] contains helm charts automatically deploying Free5GC with UERANSIM and My5G-RANTester. However, as it is solely a repository, it can be considered only as a supplementary tool that does not allow comparing different implementations of 5GC.

Several works in the literature focus on offering tools and solutions that facilitate the automated deployment and orchestration of 5G mobile networks, albeit not specifically targeting automation itself. For example, in [17] authors propose the architecture enabling topology discovery in 5G multi-tenant environment. It considers networks virtualization as the main driving factor. The process of topology discovery outlined in [17] is extensive, covering various network layers and components. Thus, it may provide a significant input for the automated orchestration. In a recent study [18], an interesting testbed was introduced. The authors integrated the emulation of the 5G radio access network and mobile core network with the simulation of user equipment. Their approach utilized container-based deployment and primarily emphasized the generation of realistic traffic patterns based on pre-collected data. The objective of this research was to facilitate future experiments conducted on 5G infrastructure. While the study offers a valuable dataset and proposes a framework for obtaining such datasets, which can be instrumental in validating and assessing automation processes, it does not directly provide solutions for automation. Chun et al. [19] recognized the potential benefits of orchestrating Kubernetes with 5G network functions as a means to automate the process. In their study, they presented several proposed enhancements for Kubernetes that have the potential to enhance the performance of the 5G core network. These contributions offer valuable insights and pave the way for automating 5G network deployments. However, it is important to note that the study does not specifically investigate the implementations of 5G networks or provide a comprehensive performance evaluation. Furthermore, in [20] authors propose a Machine Learning-based approach to predict the

number of selected network functions to efficiently handle varying number of end users registrations in the mobile core network. Solutions like that should be considered as a source of information triggering automation processes and tools.

Table 1: Summary of related works.

Work	Performance assessment		Automation aspects of 5G core deployment		
	User plane	Control plane	Tools, surveys	Original solution	Performance assessment
[6]	✓				
[7]	✓				
[8]	✓				
[9]	✓				
[10]	✓	✓			
[11]	✓	✓			
[12]	✓	✓			
[13]	✓	✓			
[14]			✓		
[15]				✓	
[16]				✓	
[17]			✓		
[18]			✓		
[19]			✓		
[20]		✓	✓	✓	
This paper	✓	✓	✓	✓	✓

Table 1 qualitatively compares the contribution of this paper with respect to the state-of-the-art. Based on the content of Table 1, it is possible to highlight the originality of our work. Works [6], [7], [8] and [9] focus mostly on the user plane performance of 5G networks. The study in [6] compares NSA and SA architectures, while the paper [7] compares 4G and 5G open-source deployments with a commercial network. Similarly, the study in [8] analyzes transmission parameters in the 5G indoor testbed, while [9] considers software engineering aspects. On the other hand, [10], [11], [12] and the recent study [13] are focused on the assessment of the 5G core network implementations and deployments including control plane aspects. The study in [10] provides some theoretical evaluations and discussions, [11] evaluates

conformance and robustness of NAS and NGAP protocols implementation, while [12] and [13] are devoted to the performance assessment and comparing 5G core network implementations. However, none of these works address the automation process, which is the primary focus of this paper.

On the other hand, the survey [14] presents tools that can be used to build a framework for automating the deployment and orchestration of 5G mobile networks. In our work, we select tools and provide a Proof-of-Concept of such a framework. Work by Wiranata et al. [15] and repository [16] propose approaches for automating the deployment of a 5G network, but do not discuss SA 5G network deployment and lack an in-depth performance assessment of the automation process comparing different implementations of the 5G core network. Therefore, our work contributes by filling this gap. Finally, works [17], [18], [19] and [20] do not propose solutions on automation but rather discuss tools that can facilitate the automated deployment and orchestration of 5G mobile networks, which is the aim of our work.

Based on the comparison presented above and summarized in Table 1, the following conclusions can be drawn. Our work is the first to propose an original solution for automated deployment of a standalone 5G core network. What is important is that the processes and tools presented in this work enable fully automated deployment in a public cloud infrastructure. Similar contributions were not provided in any previous works. Furthermore, we provide a unique efficiency assessment of the automation process for the deployment of 5G networks in both local and cloud environments, and augment it with 5G data and control plane performance assessment. Finally, we propose unique enhancements that reduced the deployment time for one of the 5G core network implementations.

Automation of the deployment process is strongly related to the service orchestration, for which a dedicated framework and tools were proposed. Service orchestrators are frameworks that facilitate the processes of designing, creating, delivering, monitoring and managing of end-to-end IT services. With the introduction of the NFV approach, service orchestrators are responsible for, among other things, deploying and interconnecting virtual network functions, managing the lifecycle of NFs, autoscaling, and fault management. These activities are denoted as network function orchestration and service orchestrators are capable to perform them also in public clouds. As a result, service and network function orchestrators are an important component in ensuring automated service provisioning, applicable also in the domain of virtualized 5G networks. The aim of our work is not to compete with or

replace orchestration frameworks, but rather to propose a solution for a specific use case, which is the automated deployment of the 5G core network. Our contribution comprises additional activities that must be performed, e.g., preparing specific virtual machines, selecting proper configuration for 5G network functions, and modifying scripts to exclude a need for human intervention. Our contribution should be considered as a supplement to the service and network function orchestrators. Service orchestrators are not intended to create and configure public cloud infrastructure; instead, their primary role is to deploy network functions onto pre-existing cloud resources.

The problem of network function orchestration is widely addressed in the literature. Thus, there exists not only a tremendous number of transactional papers proposing novel solutions, but also a significant number of surveys. The latter often propose handy taxonomy and summarizing conclusions that facilitate getting familiar with the current state-of-the-art. Thus, instead of providing a superficial survey, we suggest referring to the existing comprehensive works [21, 22, 23, 24].

3. Background

In this section, we provide some background regarding the 5G networks to facilitate replicating our original environment and understanding of the experiments. We also indicate challenges and limitations regarding the proposed automated 5G core network orchestration.

3.1. 5G network architecture

5G networks are usually perceived from the perspective of mobile technology. However, 5G consists of the two crucial components that coexist together: 5G New Radio (NR) and the core network. In our work we focus on the core part, however, we provide also brief summary on the radio component.

3.1.1. 5G New Radio Access Network

Radio Access Network (RAN) is a part of the communication system that connects the wireless equipment with the wired core network. RAN is responsible for maintaining the frequency spectrum to assure efficient usage and satisfy the quality of service. RAN in 5G is also often called NR and it consists of multiple base stations. A base station in 5G is called Next Generation NodeB (gNB). The gNB transmits and receives radio signals from the

User Equipment (UE) and then exchanges them with the core network. gNB handles both control plane and user plane traffic. NR uses numerous features to improve bandwidth and data rates. It supports the massive multiple-input multiple-output (MIMO), a technology that allows to transmit and receive the data over multiple antennas. Additionally, 5G heavily uses beamforming to improve spectral efficiency. Narrow beams are sent directly to the receiver, ensuring a better signal-to-noise ratio and thus providing higher bandwidth. Besides, gNB can be split into gNB Distributed Unit (DU) and gNB Centralized Unit (CU) [25]. DU delivers features of the physical layer and Medium Access Control (MAC), while CU takes care of mobility management and radio resource allocation. Moreover, DU supports devices within one cell. Multiple DUs are then aggregated and controlled by a single CU. DU can be placed closer to the end-user, improving the latency. On the other hand, CU can be deployed in the data center on more powerful hardware [26]. The division of CU and DU improves the flexibility of the network and customization options for various applications.

3.1.2. 5G core network

The core network provides the backbone connection between RAN and the external network. It provides control and user plane connectivity for the subscribers. With the higher data rate demands and rising requirements for the networks, the traditional approach with a monolithic core network faces numerous challenges in terms of flexibility and scalability. 5G core leverages Network Function Virtualization (NFV), a technology that allows for the deployment of particular network functions independently on commodity hardware. NFV usage enables Control and User Plane Separation (CUPS) as a step to improving the flexibility of the network. For example, we can independently scale the user plane without a need to interfere with the control plane. Diverse 5G use-cases require various functionalities and performance levels. 5G tailors the network to the customer's needs by leveraging network slicing. It allows building a set of logical networks on top of the physical equipment. Each logical network can consist of different components that support particular applications. Moreover, slices can be easily scaled up/down depending on the current number of users. Network slicing also improves the reliability of the network. Slices are isolated from each other, which means that over utilization of one component will not influence another application deployed in a different slice. The problem is even more challenging as core network deployments and slice provisioning must be

done on-demand in B5G and 6G technologies to support automated network provisioning. Therefore, automated 5G core network orchestration proposed in this work is an inevitable part of these technologies and must be mature enough to meet these challenges.

3.1.3. Network Function Virtualization of 5G Network Functions

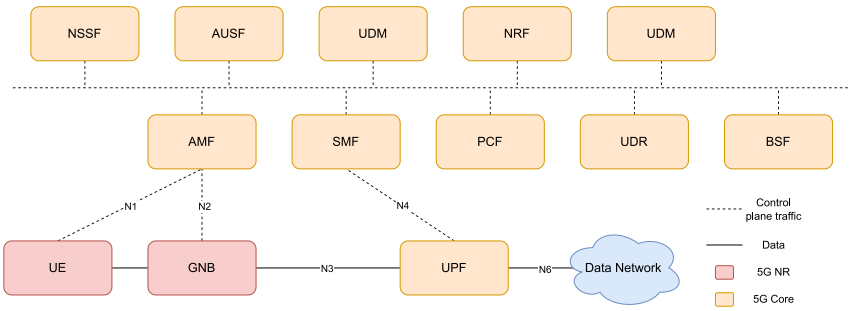


Figure 1: Architecture of 5G core network.

The 5G core network implements Service-Based Architecture (SBA) that allows providing a control plane in the form of interconnected network functions, which are independent of each other. These functions interact to provide crucial 5G capabilities and can be deployed in a centralized (monolithic) architecture or distributed across several physical and virtual machines. The modularity enabled by SBA can be leveraged in the cloud-native deployments of the 5G core, either in distributed or centralized form. Furthermore, network functions can be deployed on any hardware platfrom. This concept enables the 5G core to be rolled out in the virtualized environment as a microservice-based application. Such deployments can easily take advantage of load balancing and scale resources according to the needs [27]. These properties, despite providing obvious advantages, also impose challenging requirements on the deployment process. It is crucial to provide an automated way of deployment. The automation can be especially challenging and beneficial in the case of distributed cloud-based deployments. For these architectures, the orchestration of all distributed components should also be automated. Function separation also impacts upgrade time, allowing NFV upgrades to be performed frequently without any user disruption. This is because each function can be upgraded separately, and new versions can

be carefully prepared without affecting the current version. Additionally, the implementation of NFV enables 5G core networks to support network slicing.

Figure 1 depicts the 5G network architecture, including core network and RAN functions and connections between them. Orange and red components belong to core and RAN parts, respectively. The blue component represents an external network, typically the Internet or corporate network. More details can be found in the specification [28].

3.2. Open-source implementations of 5G network

The most popular and widely used implementations of 5G core network (Free5GC and Open5GS) are described and compared in this section. To properly test the 5G core, the radio part is also required, we leveraged UERANSIM tool [29] for this purpose.

3.2.1. Free5GC

An open-source implementation of a 5G core network. It is written in the Go language and implements Release 15 and some parts of the Release 16. Moreover, it assumes the implementation of CUPS and SBA. Free5GC requires the Linux kernel in versions 5.0.x (end of life) or 5.4.x. Free5GC provides satisfactory documentation and offers a dedicated forum for community. The authors embedded a couple of tests into the source code to verify core network deployment. Moreover, a paid membership is also offered to assure priority support and early access. Free5GC does not deliver a containerized version of the software, but there are community projects enabling this feature.

3.2.2. Open5GS

Open5GS is an open-source 5G core network written in C language. It implements Release 17 of 5G standard. This core implementation follows the most important principles of the 5G networks. It implements CUPS and SBA [30] and following network functions: Session Management Function (SMF), User Plane Function (UPF), Policy Control Function (PCF), Network Repository Function (NRF), Authentication Server Function (AUSF), Binding Support Function (BSF), Unified Data Management (UDM), Unified Data Repository (UDR), Access and Mobility Management Function (AMF), Network Slice Selection Function (NSSF). Open5GS facilitates distributed cloud-native deployments by providing Docker images for all available network functions. It is a significant advantage in the context of automated

mobile core network orchestration. Additionally, it adopts support for IPv6, multiple PDU sessions, and Voice over NR. On the other hand, it does not include roaming capabilities. It is worth mentioning that Open5GS maintains extensive documentation that includes detailed platform-specific installation guides, troubleshooting steps, and community articles.

In the Table 2 we present our original comparison of both 5G core network implementations. We especially indicate if the tool facilitates automation and orchestration of distributed deployment. The conclusion is that Open5GS is a more mature implementation in most of the aspects, including automation. Thus, it is a better candidate when considering B5G and 6G in the future, as the software will, more probably, support upcoming releases.

Table 2: The comparison of Open5GS and Free5GC.

Property	**Free5GC**	**Open5GS**
Documentation	Basic, sufficient to start using the software	Extensive, covering multiple scenarios
Supported platforms	Ubuntu, other Linux distributions that support kernel 5.0.x or 5.4.x	Debian/Ubuntu, CentOS, MacOSX(Intel and Apple Silicon), FreeBSD, Alpine
Additional requirements	gtp5g, a customized Linux kernel module	—
Implemented functions	SMF, UPF, PCF, NRF, AUSF, UDM, UDR, AMF, NSSF	SMF, UPF, PCF, NRF, SCP, AUSF, UDM, UDR, AMF, NSSF, BSF
Community and tests	Community forum, tests embedded	Community support, additional materials
Facilitates automation and orchestration	**Only community provides containerized version**	**Built-in containerized implementation**

3.2.3. UERANSIM

UERANSIM is an open-source UE and gNB implementation written in C++ language. UERANSIM simulates a 5G mobile phone and a base station [29] and realizes the NR functionalities. UERANSIM delivers three interfaces that are crucial in communication between RAN and core:

1. Control Interface (between RAN and AMF)
2. User Interface (between RAN and UPF)
3. Radio Interface (between UE and RAN)

UE also implements initial and periodic registrations. The term initial refers to a process during which UE, after being powered on, registers in the 5G core

network. Periodic registration is used by UE to inform the core that, despite recent inactivity, UE is still up and ready to transmit data [31]. UERANSIM supports the authorization of UE based on Authentication and Key Management procedure, PDU session establishment and release. gNB implements PDU Session Resource setup and release. Additionally, it provides both UE Context modification and release. However, it does not support PDU session authorization and its modification as well as IPv6. UERANSIM is compatible with multiple 5G core implementations, including both Open5GS and Free5GC.

4. The proposed automation framework for local testbed

In this section, we present the Proof-of-Concept testbed for the automated virtualized deployment of a core mobile network. Section 4.1 provides details on the configuration of the testbed and research environment. Sections 4.2 and 4.3 describe how we pre-prepared and configured virtual machines and Kubernetes cluster, respectively. Finally, the importance of Helm charts and Ansible playbooks in the context of automation is described in Sections 4.4 and 4.5, respectively. The same sections also provide details on our implementation of these components, which is an original contribution of this article.

4.1. Testbed configuration

HPE physical server is used as a foundation of the whole setup. It includes 6 CPUs x Intel(R) Xeon(R) E-2236 CPU @ 3.40GHz together with 64 GB of RAM. In order to establish a suitable environment, the virtual machine is deployed using ESXi, a hypervisor developed by VMware [32]. Ubuntu 20.04 is a base operating system for all deployments and tests. It is selected due to proven support the Open5GS project [33] and also because Free5GC authors provide tests of their software against Ubuntu [34]. These 5G core implementations are deployed on a virtual machine together with a UERANSIM. Each virtual machine is granted 6 virtual cores and 4 GB of RAM. VMs are connected to the external network via VMXNET Generation 3 (VMXNET3) interfaces. VMXNET3 is a para-virtualized Network Interface Card (NIC). Figure 2 presents an overview of the environment, where Open5GS, Free5GC and UERANSIM building blocks represent a set of containers hosting 5G network functions (every function in a separate container).

Additionally, we equipped the VMs with Docker, Kubernetes, Helm, and Ansible tools, which were utilized to automate the deployment process. Docker was used as software that divides a monolithic application into several containers by configuring and building them. These containers were deployed with the use of Kubernetes, which further also ensures connectivity between them. To accelerate deployment, we leveraged Helm charts and adapted them to the underlying network conditions within the ESXi host. This approach enabled us to deploy Free5GC and Open5GS within seconds. The cluster formation, application deployment, and Helm chart deployment were achieved using original Ansible playbooks proposed as a part of this Proof-of-Concept [5]. These efforts significantly reduced the time needed to prepare the environment from hours to minutes.

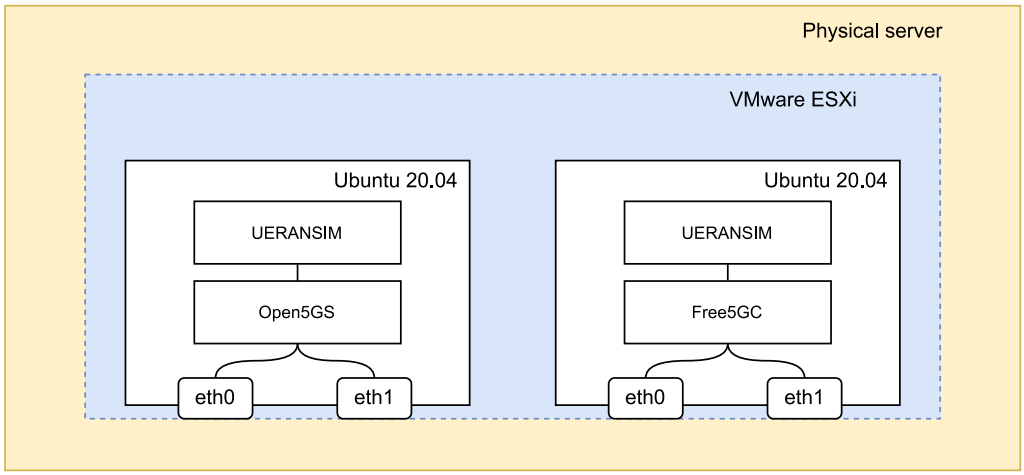


Figure 2: General overview of an environment to conduct experiments. Open5GS, Free5GC and UERANSIM are deployed in the form of containers, each hosting a separate network function.

In the following sections, we present our proposed automated orchestration for virtualized mobile core networks. The main aim is to simplify and speed up the deployment to meet the requirements of mobile core deployments. Consecutive subsections provide details about virtual machine image preparation, automation tools usage, and cluster configuration. It also presents troubleshooting required during the development and configuration phase to facilitate future works in similar environment.

4.2. Virtual Machine

We prepared a dedicated Virtual Machine image. The base operating system was enriched by dependencies that allow running both 5G core implementations within seconds. The image uses a specific kernel version – 5.4.0-58-generic because the gtp5g kernel module used by Free5GC developers was built and tested only against 5.0.x and 5.4.x kernel versions [35]. Due to Free5GC constraints, the Go language libraries were preinstalled. A virtual machine also includes additional tools crucial for automation – Docker, Kubernetes, Ansible, and Helm. Moreover, it contains pre-configuration of the virtual machine interfaces, repositories with stable versions of both 5G core implementations, and Ansible playbooks.

4.3. Kubernetes cluster

Kubernetes cluster is the key component of the proposed Proof-of-Concept. It orchestrates virtualized mobile core in the form of containers that contain images of particular 5G network functions. Kubernetes takes care of healing unresponsive containers by automatically deploying new ones. It may also perform automatic horizontal scaling. Moreover, Kubernetes also provides the networking layer of the setup by allocating appropriate IP addresses for pods and containers. A single-node cluster with all NFs deployed on the *master node* is used. Every NF container is deployed in a single pod, however, instance scaling up can be done effortlessly if needed.

4.4. Helm charts

Helm charts are the foundation of the rapid deployment of any microservice-based application. In this deployment, they provide a quick and repetitive way of setting up a containerized implementation of the 5G core network. Thus, Helm charts directly address the needs of future B5G and 6G deployments. Helm charts define and manage multi-container applications to be deployed on the Kubernetes cluster. It allows running the whole virtualized mobile core network using just one command. During tests, we used two different Helm charts:

- Towards5G-Helm – an open-source project providing on-click 5G core network and RAN Kubernetes deployment based on Free5GC and UERANSIM [36],
- Openverso Charts – an open-source project providing charts for Open5GS deployment [37].

As a contribution of our work, we made numerous configuration changes to the abovementioned Helm charts. The aim of these modifications is to allow deploying Open5GS and Free5GC without a need for human intervention. Modifications included test specific 5G core network configuration adjustments. Moreover, we also added changes required by the ESXi environment, for example, adapting network-specific elements to the charts. Charts prepared in this way are ready to ensure fully automated orchestration on the Kubernetes cluster.

4.5. Ansible playbooks

Ansible playbooks play an important role in the automation process and are available in the public repository [5]. They speed up the deployment process of the whole core network by executing a set of commands on the remote system. Playbooks are responsible for managing and deploying all configuration items, which can not be embedded into the virtual machine's image and would typically have to be configured manually. Thus, a tool similar to the Ansible is an inevitable part of the automated orchestration in the context of mobile core networks.

As an original contribution of this work, we developed four Ansible playbooks that facilitate 5G core network deployment (two playbooks for each mobile core implementation). The first playbook creates a Kubernetes cluster on the virtual machine, edits Helm charts, and deploys the 5G core network using the charts. The second one sets up the UERANSIM subscriber. Selected, the most important, parts of the proposed playbooks are explained below:

- We properly prepared the operating system to run Kubernetes cluster in terms of swap memory, iptable and arp tables, and installation of Kubernetes with kubelet, kubeadm, kubectl tools. We also automated installation of Helm.

```
- name: Install helm
    unarchive:
      src: "https://get.helm.sh/helm-v{{ helm_version }}
      -linux-amd64.tar.gz"
      dest: /usr/local/bin
      extra_opts: "--strip-components=1"
      owner: root
      group: root
      mode: 0755
      remote_src: true
```

- We ensured networking layer between all components with the use of Flannel Container Network Interface (CNI) [38] together with Multus CNI [39]. We previously automated the download and installation of both plugins.

```
- name: "Apply Kubernetes routing - Flannel and Multus CNI"
  shell: |
    kubectl apply -f flannel.yaml
    kubectl taint nodes $(hostname) node-role.kubernetes.io/
    master:NoSchedule-
    kubectl create ns open5gs
    cd multus-cni
    cat ./deployments/multus-daemonset.yml | kubectl apply -f -
```

- Automatically deploying the complete and fully operational virtualized mobile core network using prepared Helm charts.

```
- name: "Create dir for Helm charts"
    ansible.builtin.file:
      path: /opt/charts
      state: directory
      owner: root
      group: root
      mode: 0755

  - name: "Get Helm chart for openverso"
    ansible.builtin.git:
      repo: https://github.com/Gradiant/5g-charts.git
      dest: /opt/charts/openverso-charts

- name: "Deploy Open5GS Core Network"
  shell: |
    cd ~/openverso-charts/charts/open5gs/
    helm dependency update
    cd ~/openverso-charts/charts/
    helm -n open5gs install v1 ./open5gs/
```

- To propose complete Proof-of-Concept the automation process covers also radio part simulated by the UERANSIM. The playbook determines IP address of WEBUI service required to register a mobile subscriber. Without our proposal, the user would need to find the WEBUI pod, investigate its networking details and then proceed with registration.

```
- name: "Find IP address of WEBUI and ask user to register the
subscriber"
```

```
shell: |
  WEBUI=kubectl get pods -n open5gs | grep web | grep -oE ^.*\-.?
  WEBIP=kubectl describe pod $WEBUI -n open5gs |  grep -oE
  "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" | tail -1
  echo "Go to $WEBIP:3000, log in with admin/1423 and
  register your UE(s)"
```

- The gNB requires the IP address of AMF to register to the network. However, as this address is assigned during deployment it should be automatically injected to the gNB chart. To achieve that we also proposed a playbook.

```
- name: "Find IP address of AMF and supply \gls{gNB} chart with it"
  shell: |
    AMF_pod=kubectl get pods -n open5gs | grep amf | grep -oE ^.*\-.?
    AMF_IP=kubectl describe pod $AMF_pod -n open5gs |  grep -oE
    "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" | tail -1
    cd ~/openverso-charts/charts/ueransim/resources
    sed -i "s/\(address:\).*/\1 $AMF_IP/g" gnb.yaml
```

5. Evaluation results

Based on the presented Proof-of-Concept, we further verify performance of both implementations of virtualized mobile core networks: Free5GC and Open5GS. Each subsection begins with a detailed explanation of the setup. For this study, we conducted tests using a single UE communicating through an isolated PDU session. This approach ensures that the results are not influenced by external factors. We conducted ten consecutive measurements for each test, followed by statistical analyses to calculate and present 95% confidence intervals.

5.1. User plane delay

User plane delay is one of the most crucial metrics from the end-user perspective, as it pertains to the transmission of traffic in the user plane, crucial for data network operations. It directly impacts the delay experienced by customers, which is especially critical given the emerging applications offered through B5G and 6G networks. Therefore, the primary objective is to minimize delay as much as possible for selected applications. In this context, we measured the delay introduced by the UPF during data transfer from the UE to the external network. To ensure reliable measurements,

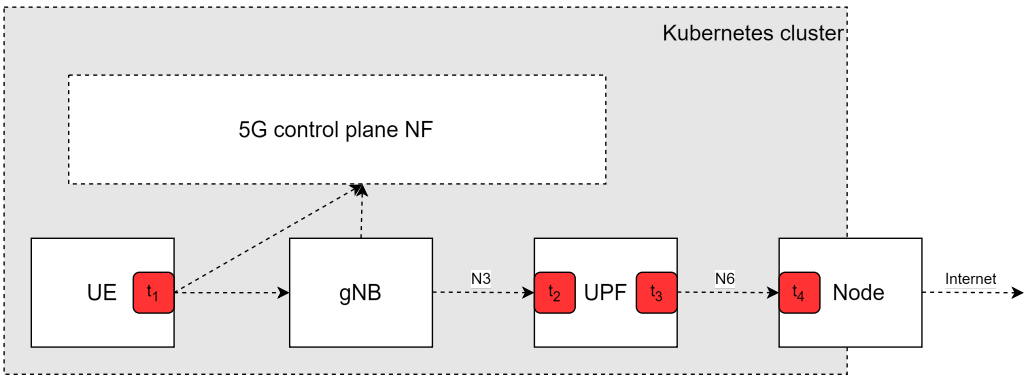
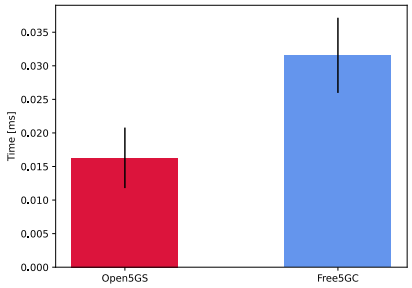


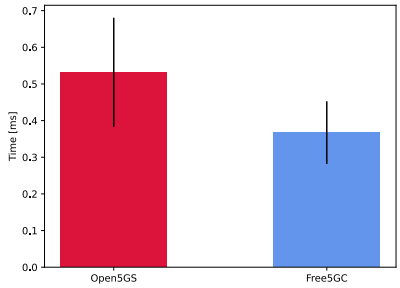
Figure 3: Topology with capture points for the purpose of user plane delay test.

packets were captured on four interfaces highlighted in red and labeled as t_1, t_2, t_3 and t_4 in Figure 3. The *Node* component in Figure 3 represents a virtual machine on which Kubernetes cluster was deployed (as detailed in Figure 2), thus, it should be considered as a point at which traffic leaves 5G core network deployment.

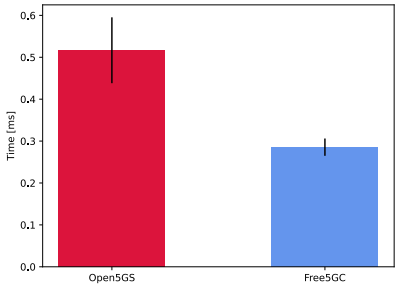
Figure 4a presents uplink delay reported by the UPFs of Free5GC and Open5GS tools. The single result was obtained by analyzing the time difference between packet arrival time and the time when a packet reached the egress interface of UPF (difference between time measured in t_3 - t_2). It can be observed that average delay introduced by Free5GC UPF is twice higher than the analogous delay in case of Open5GS. These results depend on the particular implementation and software stack (including programming language) being used by both tools.



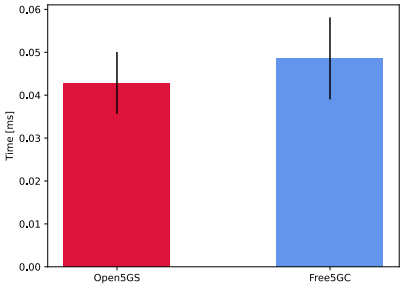
(a) Time difference between packet sending out of UPF egress (t_3) and packet arrival on UPF ingress (t_2) interface.



(b) Time difference between packet arrival on node ingress (t_4) and sending the packet out of UE egress (t_1) interface.



(c) Time difference between packet arrival on UPF egress (t_3) and sending the packet out of UE egress (t_1) interface.



(d) Time difference between packet arrival on node ingress (t_4) and packet arrival on UPF ingress (t_2) interface.

Figure 4: User plane delay measurements for Open5GS and Free5GC based on the time differences between various actions.

However, one must note that packets are captured after they are received on the ingress interface. It does not include the delay introduced by the kernel and the egress interface. To verify which 5G core implementation introduces a lower delay, we captured packets on the UE egress interface and node ingress interface (difference between time measured in t_4 - t_1). Figure 4b depicts the delay reported by the next-hop devices and can be considered as an end to end latency in the user plane of the 5G core network. This test reveals that, Open5GS imposes the delay that on average is slightly higher than the one introduced by Free5GC. However, taking into account confidence intervals, results can be considered as statistically equal. This conclusion is contrary to previous observations. The absolute delay value

can't be compared with previous measurements, as this test includes more components.

We performed additional analysis of the achieved values to explore which part (ingress or egress) of UPF is mostly responsible for introducing the delay. For this purpose, Figures 4c and 4d should be analyzed together. Figure 4c depicts the time difference between packet arrival on UPF egress interface and sending the packet out of UE egress interface (difference between time measured in t_3 - t_1). Figure 4d shows time difference between packet arrival on node ingress interface and packet arrival on UPF ingress interface (difference between time measured in t_4 - t_2). It can be observed that packet processing takes ten times longer on the path from UE to UPF than between UPF and node. This is expected behavior as there is also gNB involved in packet forwarding between UE and UPF.

Figure 4c shows that Open5GS introduced on average approximately 0.2 ms more delay on the path between UE and UPF than Free5GC. Please also note that UE and gNB modules are the same for Open5GS and Free5GC, thus, any differences originate from the implementation of the UPF component. On the path UPF – node (shown in Figure 4d), both implementations performed equally. Thus, based on the results presented in Figures 4c and 4d, it can be concluded that Open5GS core network processes packets internally faster than Free5GC. However, it takes Open5GS more time to deliver a packet from the ingress interface to the main computing component. These measurements also allowed us to understand better results seen in Figure 4a.

5.2. Throughput

Another critical indicator of the user plane performance is throughput. In addition to the previous setup, we provisioned a supplementary virtual machine with Ubuntu. This Virtual Machine (VM) was used solely for measurement purposes, as shown in Figure 5.

The experiment is conducted using iPerf3 that works in the client-server architecture. In this test, the additional VM provides the server role while UE acts as a client. During the first attempt, all parameters are set to default. Traffic is sent in both directions between the machines.

Figure 6 depicts throughput achieved during experiments. TCP was used as a transport protocol, with 128 KB as a datagram size. During tests, Open5GS achieved around 375 Mb/s on average. Gabilondo et. al. in work [12] also studied the throughput of various 5G cores, and for Open5GS the reported throughput was two times lower compared to our results. The

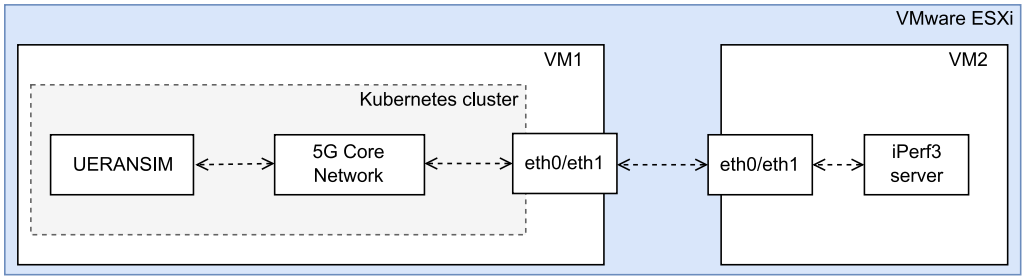


Figure 5: Topology used to measure throughput.

most significant difference in our setups is the type of UE/RAN device. We used UERANSIM, while Gabilondo et al. used a physical wireless UE device. Comparing Free5GC and Open5GS it can be observed that Open5GS achieves, on average, around 3.5 times higher throughput than Free5GC.

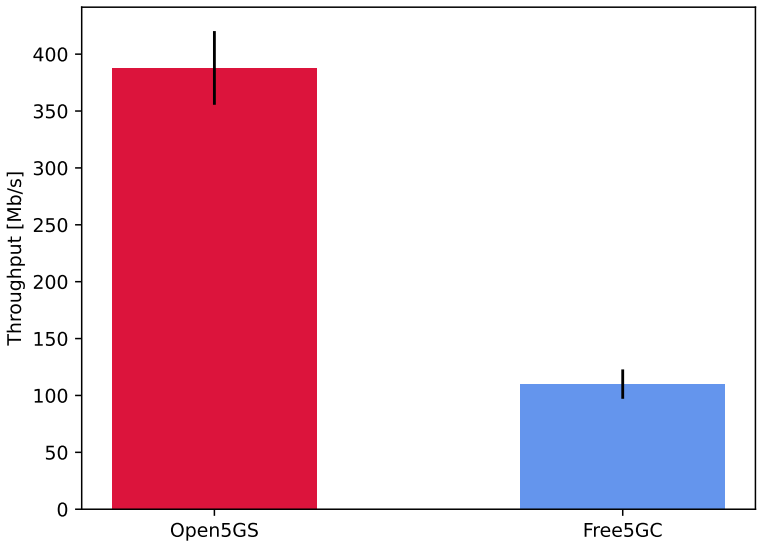


Figure 6: Throughput achieved by one UE for both 5G core implementations.

5.3. Memory usage

Such a significant difference in terms of throughput initiated further comparative studies on memory and CPU usage. For that purpose, we installed

the Kubernetes metrics-server – the component responsible for collecting metrics from the individual containers. Figures 7 and 8 illustrate memory consumption during the throughput test. One square on the images corresponds to one mebibyte (1024 * 1024 bytes).

Both tests revealed that the most utilized is UPF container, what is reasonable as the measurements were done during network saturation. Other functions consume significantly less memory, and the amount is roughly equal for each function (with the except for SMF function in Open5GS). Furthermore, the difference in overall memory usage can be observed – 461 Mi for Open5GS, while Free5GC uses only 103 Mi. In general, every single network function consumes more memory in the implementation provided by Open5GS than by Free5GC. Simultaneously, the proportions of memory consumed by the network functions is the same for both tools. Therefore, the difference is most probably the result of the implementation method, software stack and memory management.

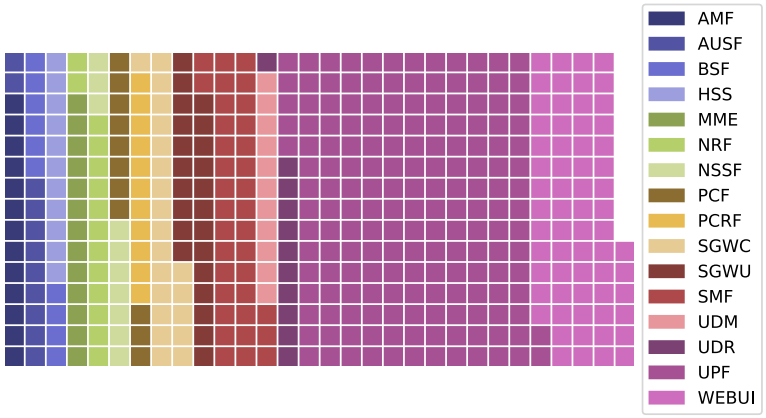


Figure 7: Memory allocation for Open5GS. One square on the image corresponds to one mebibyte (1024 * 1024 bytes).

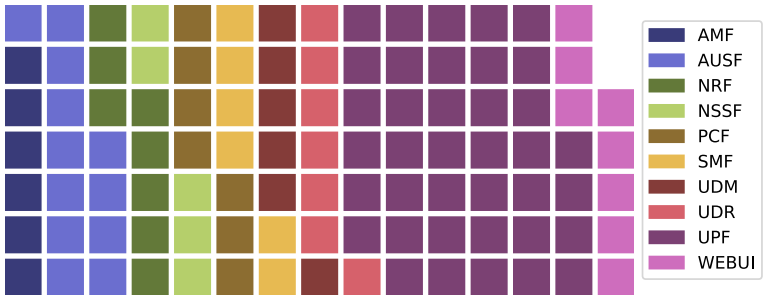


Figure 8: Memory allocation for Free5GC. One square on the image corresponds to one mebibyte (1024 * 1024 bytes).

5.4. Study on scalability

Due to the differences observed in the previous experiment, we conduct some additional studies on scalability of Free5GC trying to improve its performance. Scalability is also a critical feasibility indicator in relation to mobile core networks. We removed any restrictions on CPU and memory applied to Free5GC by default configuration of Towards5GS-helm charts which utilize *limits* Kubernetes feature against the UPF container to provide it with 0.5 of CPU unit and 500 mebibytes of memory. Removed limitations are especially important in the context of the UPF, being responsible for forwarding users' data through the 5G core network.

Figure 9 presents achieved throughput after configuration tuning. Free5GC achieved almost 1.5 more throughput than in the previous experiment, however, the difference to the Open5GS result is still noticeable (see Figure 6). Interestingly, the memory usage for Free5GC grew from 103 Mi to 124 Mi, while UPF was consuming 48 Mi (38 Mi previously). Thus, we can conclude that the customization process should be performed before any single deployment to meet the requirements.

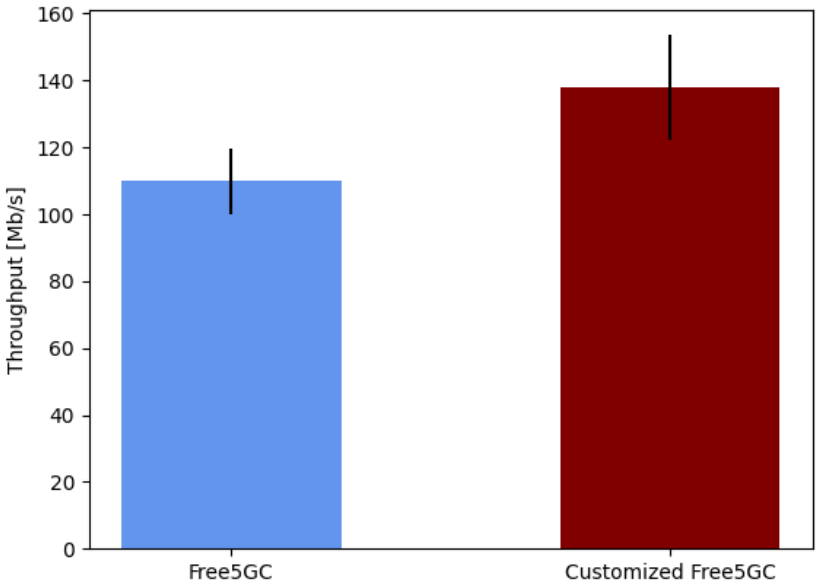


Figure 9: Throughput achieved by one UE using Free5GC implementations with default and optimized configuration.

5.5. Control plane performance

In this experiment, we focus on the performance of the control plane, in contrast to the user plane considered in all previous subsections. We will examine the UE registration procedure. For end-users, a robust registration process is critical for quickly accessing the network, a factor that becomes even more important when considering on-demand network provisioning in the case of B5G deployments. Time measurement begins with the UE's registration request and concludes when the PDU session is established. To provide comprehensive results, we measured the time needed to handle UE registration and establish a PDU session between the 5G core network and the UE. For both core deployments, we used analogous UERANSIM charts to deploy virtual UEs to ensure the credibility of the results.

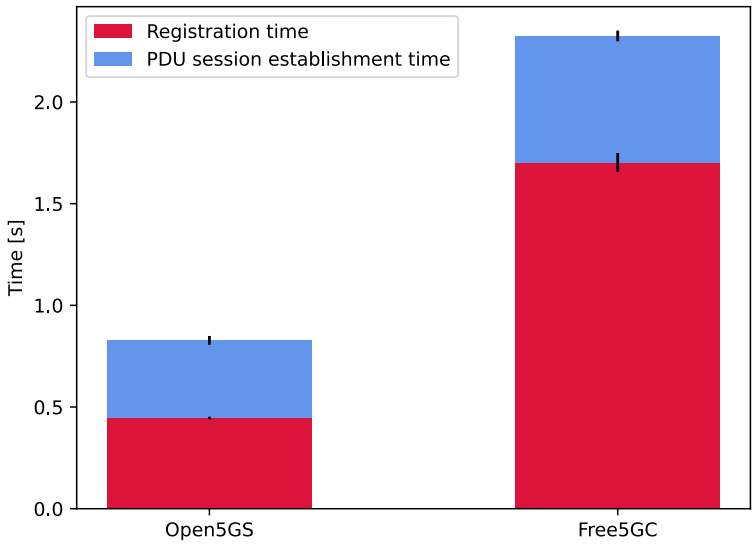


Figure 10: Time needed for UE registration and PDU session establishment.

Figure 10 depicts the registration time and PDU establishment time. Results in consecutive runs were similar and stable, causing confidence intervals to be in the sub-second domain. Open5GS was faster than Free5GC by around 1.5 s on average. The first component of the total time – the registration time, took approximately 0.45 s for Open5GS, whereas Free5GC registered the subscriber in more than 1.5 s. This significant difference can be already noticed by the end user. Open5GS also offers a faster session establishment time than Free5GC, by only approximately 0.1 s. It is worth noticing that the registration time to PDU session establishment time ratio varies in both implementations. For Open5GS, registration time was only 28% longer than PDU establishment time, whereas Free5GC registration time was almost four times longer.

5.6. Performance evaluation of the automated 5G core network orchestration

Mobile core networks are designed with the assumption that particular network functions will be virtualized. Both Free5GC and Open5GS implement this concept. The programmatic character of the network allows running it on the cloud, deploy it on-demand, and scale it with the use of

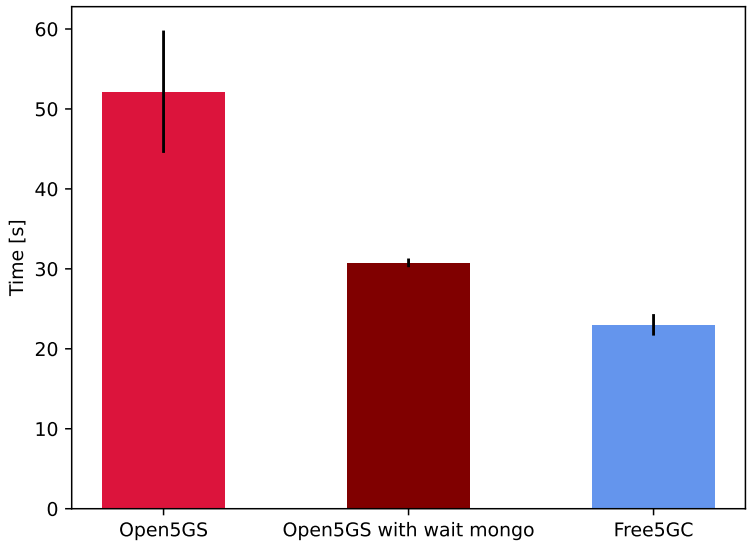


Figure 11: Time needed to automatically deploy virtualized 5G core network.

automated orchestration. Thus, the time needed to set up the network on-demand is an important parameter for emerging use cases.

This experiment verifies the time needed to get the fully operational 5G core network. Time was measured since applying the Helm command, starting the deployment, until all core network pods are up and running.

The initial set of experiments revealed that Open5GS requires approximately 50 seconds to create, while the Free5GC network can be deployed in just around 25 seconds. The origins of this behavior come from the Helm chart implementations. Namely, some network functions in 5G networks need to be created before others, for example, the database needs to be created earlier than AMF, which is responsible for the registration of subscribers. Default Kubernetes restart policy will try to reschedule pods with an exponential back-off delay (10 s, 20 s, 40 s, etc.) [40]. The deployment of Open5GS utilizes this mechanism, which causes additional delay trying to reschedule some containers accordingly to the back-off time. In every Open5GS deployment, containers: HSS, PCF, PCRF, and UDR restarted twice, what strongly exceeds the deployment time. Helm charts for Free5GC leverage a couple of init containers deployed prior to a container with desired

network function: *wait mongo* and *wait NRF*. These containers are *busybox* instances, which are executables that incorporate Unix utilities [41]. Both containers work in a loop checking, respectively, if the MongoDB and NRF pods are running. If not, containers wait 2 seconds before the next iteration. Dependent network functions are deployed once *wait mongo* and *Wait NRF* detect that MongoDB and NRF are up and running, respectively.

Similar mechanism is not implemented (besides *wait NRF* container in BSF) in Openverso charts for Open5GS. Thus, we decided to implement an original enhancement: the *wait mongo* container for HSS, PCF, PCRF, and UDR pods in Open5GS deployment. Our contribution created the same pods' startup conditions. Hence, the comparison of the deployment time of Open5GS and Free5GC is reliable. Figure 11 depicts deployment time for three setups: Open5GS, Open5GS with our original enhancement *wait mongo*, and Free5GC. Our implementation of the init containers reduced the Open5GS deployment time by 40% on average. Despite proposed improvements to Open5GS charts, Free5GC deployment is still around 16% faster on average.

5.7. Discussion

The results of the tests conducted in our work offer insights that are valuable from the broader perspective of future automated orchestration for mobile core deployments. In summary, Open5GS seems to be more suitable for commercial use. It delivers significantly greater throughput and better support for the automation. Moreover, Open5GS supports higher amount of operating systems and does not introduce any kernel limitations. Our experiments revealed that UPF in both Open5GS and Free5GC introduces a similar delay. In the stable core network environment, Open5GS ensures faster registration for UE and PDU establishment time. However, Free5GC is more suitable for educational and testing environments, provisioning the network faster than Open5GS and consuming less memory.

When considering automation, it is crucial to examine and understand the architecture specific to deployment while analyzing deployment time. Moreover, each automation tool should be customized to the specific environment. Taking into consideration deployment time and total registration time, we can observe various applications for particular 5G core implementations. Due to its shorter deployment time, Free5GC appears more suitable for educational purposes, while Open5GS prioritizes faster registration of UEs, which is crucial in a production environment.

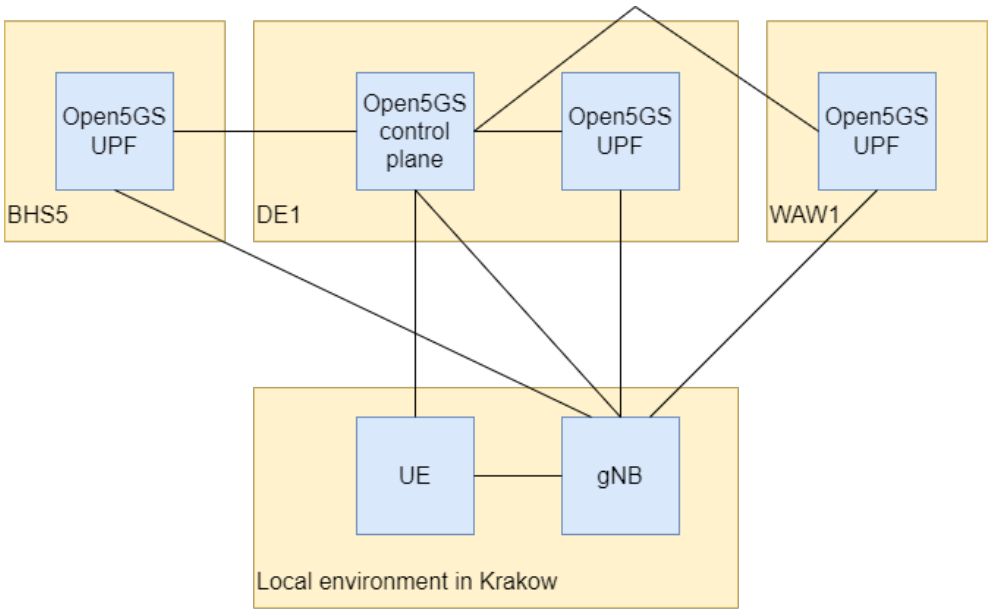


Figure 12: Architecture of cloud-based 5G core network deployment.

6. Automation of cloud-based 5G network deployment

Due to the widespread availability of cloud providers' resources, cloud-based deployment is a natural next step as an underlying infrastructure for the 5G network components. In case of the cloud, 5G functions can also be deployed in the centralized or distributed manner, in which physical and virtual machines hosting network functions can be placed in different physical locations. Automation can be especially beneficial in such distributed and complex cloud-based architectures, it should also comprise more aspects (e.g., creating infrastructure, orchestration, or ensuring connectivity).

In this section, we propose the process and necessary tools to automatically deploy a virtualized 5G core network in a cloud infrastructure. OVH Cloud was chosen because it allows access to the *Openstack* platform, enables the launch of services in many locations around the globe, and is attractively priced. Multiple UPFs are separated from the control plane and deployed in different cloud regions to ensure geographical proximity for the end-users.

Based on the discussion and results presented in Sections 3 and 5, respectively, we selected Open5GS as a more prominent candidate for a cloud-based deployment. The proposed deployment process comprises the following steps:

- creation of the cloud computing infrastructure and preparing its initial configuration (Section 6.1),
- preparing images of virtual machines (Section 6.2),
- installing and configuring a virtualized 5G network (Section 6.3).

We designed an architecture in which UPF functions (user plane of 5G core network) are separated from the rest of the 5G core functions (control plane). Furthermore, multiple UPFs are deployed in different cloud regions to ensure geographical proximity for the end-users. We utilized the following regions of the OVH Cloud infrastructure, *WAW1* - Warsaw, *DE1* - Frankfurt and *BHS5* - Beauharnois. For evaluation, we also considered a reference, centralized scenario in which the entire core of the 5G network (control and user planes) is deployed on a single virtual machine in one location (*DE1*). Similar fully automated cloud-based deployments are not currently available, even with the use of service orchestrators.

The infrastructure comprises also local resources run in Krakow with the use of Dell Inspiron 5505 laptop with an AMD Ryzen 7 4700U processor, 16 GB RAM and a 512 GB SSD. The local environment was used solely to simulate UE and gNB using UERANSIM access network simulator. Figure 12 presents the designed architecture.

In the following, we outline the necessary steps to deploy 5G core in the cloud and provide some tools and solutions enabling automation. In Section 6.4 we summarize the differences between local testbed and cloud-based deployment. We also perform some preliminary performance assessments and analyze the results in Section 6.5.

6.1. Creating and configuring cloud infrastructure

In our deployment, we utilized infrastructure of OVH Cloud. In the case of OVH, creating the infrastructure for the deployment had to be preceded by configuring the project. For this purpose, the project−setup module was created. The module is responsible for creating and configuring the network between the data centers in different geographical locations.

All resource definitions created by this module are included in the single Terraform[4] file automating the process (called Terraform module). Listing 1

[4]https://www.terraform.io/

shows the example of a resource for creating a network in the OVH Cloud project. The network has been configured in several selected regions and has been assigned VLAN ID 30, distinguishing the networks in the project.

```
resource "ovh_cloud_project_network_private" "network" {
   service_name = local.ovh_project_id
   name = "private_network_30"
   regions = ["WAW1", "DE1", "BHS5"]
   vlan_id = 30
}
```

Listing 1: Network configuration for the project

A subnet was then configured for each region. An example resource that creates a subnet for the *WAW1* region is shown in Listing 2. The entire network had a pool of addresses of 10.30.0.0/16, and the address pools selected for the configuration of the presented subnet starting from 10.30.0.2 and ending with 10.30.0.254. Parameter network_id associates the subnet to the previously created network.

```
resource "ovh_cloud_project_network_private_subnet" "subnet_waw" {
   service_name = local.ovh_project_id
   network_id = ovh_cloud_project_network_private.network.id
   start = "10.30.0.2"
   end = "10.30.0.254"
   network = "10.30.0.0/16"
   dhcp = true
   region = "WAW1"
   no_gateway = true
}
```

Listing 2: Subnet configuration for the selected region

The project—setup module is also responsible for creating a user for project management. The resource responsible for creating this user is shown in Listing 3. The configuration includes roles corresponding to user rights.

After configuring the project, the infrastructure for deployment can be created. To automate the process, we also used Terraform modules. These modules are responsible for, among others: adding an SSH key used to provide access to virtual machines, guaranteeing the placement of virtual machines on different physical machines, creating virtual machines.

The contents of the automation file is shown in Listing 4.

The variable vms, is used to configure the virtual machines being created in each region. We created a distributed infrastructure for the 5G core

```
resource "ovh_cloud_project_user" "terraform" {
   service_name = local.ovh_project_id
   description = "terraform"
   role_names = [
     "administrator",
     "authentication",
     "backup_operator",
     "compute_operator",
     "image_operator",
     "infrastructure_supervisor",
     "network_operator",
     "network_security_operator",
     "objectstore_operator",
     "volume_operator",
   ]
}
```

Listing 3: Configuration of user for project management

```
locals {
   vms = {

     DE1 = [
       {name = "open5gs-core-de", image_id = "69e11759-6988-4ad1-a533-2f4d4ae36b39",
       flavor = "b2-7", fixed_ip_v4 = "10.30.1.10"},
       {name = "open5gs-upf-de-1", image_id = "69e11759-6988-4ad1-a533-2f4d4ae36b39",
       flavor = "b2-7", fixed_ip_v4 = "10.30.1.40"},
       {name = "lb-de-1", image_id = "b754a116-dffb-4e3c-bbfe-0778f34b6304",
       flavor = "b2-7", fixed_ip_v4 = "10.30.1.11"},
     ],

     WAW1 = [
       {name = "open5gs-upf-waw-1", image_id = "f51863dd-1655-4949-b1cd-33509de1ce4e",
       flavor = "b2-7", fixed_ip_v4 = "10.30.0.13"},
     ],

     BHS5 = [
       {name = "open5gs-upf-bhs-1", image_id = "d398f2fb-387d-45f3-bda2-2f111f603e22",
       flavor = "b2-7", fixed_ip_v4 = "10.30.4.14"},
     ]
   }
}
```

Listing 4: Terraform module to automatically create a cloud infrastructure for the 5G deployment

network deployment. Namely, the UPF network function was automatically deployed in several regions. The following variables were used to define the created virtual machines:

- name - name of the virtual machine,

- image - name of the virtual machine image, used when starting the machine with the base operating system,
- image_id - image ID of the virtual machine, used when starting a machine with a pre-installed image,
- flavor - an identifier representing the specification of the virtual machine, for example the amount of available RAM[5],
- fixed_ip_v4 - static IP address assigned to the virtual machine.

A separate module defines resources in each region. An example for DE1 region is shown in Listing 5. The resource type openstack_compute_keypair_v2 was used to create the SSH key. The resource type openstack_compute_servergroup_v2 was used to create a group with the policy *anti-affinity*. To create virtual machines, the openstack_compute_instance_v2 type was used. The count configuration parameter was used to avoid repeating the same block of code for each separate machine and determines the number of resources to be created. Based on the vms local variable, the virtual machine settings were configured. Additionally, an output variable ansible_host is created, which stores the public IP addresses of the created virtual machines.

The last step is to ensure proper connectivity between the gNB and UE simulators and the 5G core network. This communication requires direct visibility between machines, and this is not possible when Internet access is provided via NAT. Additionally, communication between the base station simulator and the core network uses the SCTP protocol, which is often blocked by public cloud and Internet providers. To mitigate these obstacles we used the *WireGuard* tunnels and automated the process of establishing them with the use of ansible playbooks.

6.2. Creating images of virtual machines

Creating virtual machine images with pre-installed software facilitates and accelerates the process of automated deployment of virtualized 5G core network in a cloud infrastructure. For this purpose, we used Packer[6] tool. Listing 6 shows a fragment of a Packer configuration file for a virtual machine hosting Open5GS core network. The source block defines the machine from

[5] `https://www.ovhcloud.com/pl/public-cloud/prices/`
[6] https://www.packer.io/

```
resource "openstack_compute_keypair_v2" "piotr_keypair_de" {
   name = "piotr_keypair_region_de"
   region = "DE1"
   public_key = file("../../../ansible/files/ssh_keys/piotr.pub")
}
resource "openstack_compute_servergroup_v2" "anit-affinity-compute-group-de" {
   region = "DE1"
   name = "compute-group-anti-affinity-de"
   policies = ["anti-affinity"]
}
resource "openstack_compute_instance_v2" "instances" {
   count = length(local.vms.DE1)
   region = "DE1"
   name = local.vms.DE1[count.index].name
   image_name = try(local.vms.DE1[count.index].image, null)
   image_id = try(local.vms.DE1[count.index].image_id, null)
   flavor_name = local.vms.DE1[count.index].flavor
   key_pair = openstack_compute_keypair_v2.piotr_keypair_de.id
   security_groups = ["default"]
   stop_before_destroy = true
   scheduler_hints {
     group = openstack_compute_servergroup_v2.anit-affinity-compute-group-de.id
   }
   network {
     name = "Ext-Net"
   }
   network {
     name = "private_network_30"
     fixed_ip_v4 = local.vms.DE1[count.index].fixed_ip_v4
   }
}
output "ansible_host" {
   value = {
     for instance in openstack_compute_instance_v2.instances : instance.name =>
     "ansible_host=${instance.access_ip_v4} internal=${instance.network.1.fixed_ip_v4}"
   }
}
```

Listing 5: Resource definition for the selected region.

which the virtual machine image will be created. The image building process is based on using a running virtual machine and then configuring it using the prepared Ansible playbook file. The Ansible file selection and the variables passed to it are defined using the provisioner "ansible—local" block. All machines were configured to run the Ubuntu 20.04 operating system.

6.3. Configuration of the virtualized 5G network

To enable automated deployment of virtualized 5G core network in the cloud infrastructure, we had to modify configuration files of the AMF, UPF and SMF network functions. All default configurations can be found in the

```
source "openstack" "open5gs" {
   identity_endpoint = var.OS_AUTH_URL
   password = var.OS_PASSWORD
   region = var.OS_REGION_NAME
   tenant_name = var.OS_TENANT_NAME
   tenant_id = var.OS_TENANT_ID
   username = var.OS_USERNAME
   domain_name = "default"
   flavor = "b2-7"
   image_min_disk = 20
   image_name = "open5gs"
   source_image_name = "Ubuntu 20.04"
   ssh_username = "ubuntu"
   ssh_ip_version = 4
   ssh_handshake_attempts = 20
   networks = [var.network_id]
}

build {
   sources = [
     "source.openstack.open5gs",
   ]
   provisioner "shell" {
     inline = [
       "sudo apt-get update",
       "sudo apt-get install -y software-properties-common",
       "sudo apt-add-repository --yes --update ppa:ansible/ansible",
       "sudo apt-get install -y ansible"
     ]
   }
   provisioner "ansible-local" {
     only = ["openstack.open5gs"]
     playbook_dir = "../../ansible/"
     playbook_file = "../../ansible/playbooks/packer_open5gs.yml"
     role_paths = [".../../ansible/roles"]
     galaxy_roles_path = "../../ansible/roles"
     extra_arguments = [
       "--extra-vars",
       "ovh_region=\"${var.dc_name}\""
     ]
   }
}
```

Listing 6: A fragment of a Packer configuration file for a virtual machine hosting Open5GS core network.

Open5GS project repository[7]. We changed the value of NGAP interface IP address to make it available to the base station simulator. The local tunnel address value was used.

[7] https://github.com/open5gs/open5gs

```
logger:
     file: /var/log/open5gs/amf.log

sbi:
   server:
     no_tls: true
   client:
     no_tls: true

smf:
     sbi:
       - addr: 127.0.0.4
         port: 7777
     pfcp:
       - addr: {{ smf_address }}
     gtpc:
       - addr: {{ smf_address }}
     gtpu:
       - addr: {{ smf_address }}
     metrics:
       - addr: 127.0.0.4
         port: 9090
     subnet:
       {%- for hostname, subnets in upf_subnets.items() %}
       {% for subnet in subnets %}

       - addr: {{ subnet.addr }}
         dnn: {{ subnet.dnn }}
       {% endfor %}
       {%-endfor %}

     dns:
       - 8.8.8.8
       - 8.8.4.4
     mtu: 1400
     ctf:
       enabled: auto

upf:
   pfcp:
    # {%- for hostname, subnets in upf_subnets.items() %}

       - addr: "{{ upf_pfcp_address[hostname] }}"
         dnn: [{% for subnet in subnets %}{{subnet.dnn}}{%- if not loop.last %},
         {%- endif %}{% endfor %}]
    # {%- endfor %}
```

Listing 7: Modified SMF configuration file to enable automated deployment in cloud infrastructure.

The configuration file for the SMF is shown in Listing 7. We significantly modified it to enable the automation process. The smf section configures IP addresses for the service-based interface, PFCP interface, GTP interface,

```
upf_subnets:
   open5gs-upf-waw-1:
     - addr: 10.45.0.1/16
       subnet: 10.45.0.0/16
       dnn: internet-waw
       dev: ogstun
   open5gs-upf-bhs-1:
     - addr: 10.46.0.1/16
       subnet: 10.46.0.0/16
       dnn: internet-bhs
       dev: ogstun
   open5gs-upf-de-1:
     - addr: 10.47.0.1/16
       subnet: 10.47.0.0/16
       dnn: internet-de
       dev: ogstun

upf_pfcp_address:
   open5gs-upf-waw-1: 10.30.0.13
   open5gs-upf-bhs-1: 10.30.4.14
   open5gs-upf-de-1: 10.30.1.40
```

Listing 8: Variables used in the SMF configuration file.

and exported metrics. This section also automatically defines all available subnets along with the DNN (Data Network Name) using dedicated loops. The upf section is used to specify the UPF instances spawned in the network and what data networks they support. Communication sessions for UPF data are created using the addresses provided in this section. The variables used to generate the configuration are shown in Listing 8. Each UPF supports a different subnet and data network name depending on the region in which it was launched.

We also significantly modified the configuration file for the UPF function shown in Listing 9. Similarly, we prepared a dedicated loop to fully automate the deployment process. The template uses the same variables to generate the configuration of possible subnets that were used to generate the configuration for SMF. The upf section defines access addresses. The PFCP address is used for communication with the SMF, and user communication takes place via the GTPU address. Subnet specifies the available subnets along with data network access.

Finally, we also automated the deployment process for the UERANSIM tool comprising a base station simulator component and a user terminal sim-

```
logger:
     file: /var/log/open5gs/upf.log
sbi:
   server:
     no_tls: true
   client:
     no_tls: true

upf:
     pfcp:
       - addr: {{ upf_address_pfcp }}
     gtpu:
       - addr: {{ upf_address_gtpu }}
     subnet:
# {%- for subnet in upf_subnets[inventory_hostname] %}

       - addr: {{ subnet.addr }}
         dnn: {{ subnet.dnn }}
         dev: {{ subnet.dev }}
# {%- endfor %}
     metrics:
       - addr: 127.0.0.7
         port: 9090
```

Listing 9: Modified UPF configuration file to enable automated deployment in cloud infrastructure.

ulator component that can be run independently. The Vagrant[8] integration with Ansible was used to configure the newly created virtual machines hosting both components. For this purpose, a *playbook* was prepared, the task of which was to install the UERANIM software and properly configure the terminals. The machines were created using the vagrant up command (locally) or with the use of Docker containers (in the cloud environment). The proposed automation method facilitates the expansion of the deployment with more UEs and gNB components to easily conduct studies on the scalability of the 5G infrastructure. The terminal configuration is shown in Listing 10.

6.4. Comparison of local testbed and cloud-based deployment

Both infrastructures differ significantly in numerous aspects. Firstly, a local testbed comprises a single node. In the cloud, we deployed virtual machines distributed over physically separated regions and a physical host to take full advantage of the cloud infrastructure. Thus, the automation process, in cloud-deployment case, must comprise the configuration of cloud resources,

[8]https://www.vagrantup.com/

```yaml
supi: 'imsi-999700000000003'
mcc: '999'
mnc: '70'
protectionScheme: 0
homeNetworkPublicKey: '5a8d38864820197c3394b92613b20b91633cbd897119273bf8e4a6f4eec0a650'
homeNetworkPublicKeyId: 1
routingIndicator: '0000'
key: '465B5CE8B199B49FAA5F0A2EE238A6BC'
op: 'E8ED289DEBA952E4283B54E88E6183CA'
opType: 'OPC'
amf: '8000'
imei: '356938035643803'
imeiSv: '4370816125816151'

gnbSearchList:
   - 192.168.0.74

uacAic:
   mps: false
   mcs: false

uacAcc:
   normalClass: 0
   class11: false
   class12: false
   class13: false

sessions:
   - type: 'IPv4'
     apn: 'internet-waw'
     slice:
       sst: 1
configured-nssai:
   - sst: 1

default-nssai:
   - sst: 1
     SD: 1

integrity:
   IA1: true
   IA2: true

ciphering:
   EA1: true
   EA2: true

integrityMaxRate:
   uplink: 'full'
   downlink: 'full'
```

Listing 10: Modified UE configuration file to enable automated deployment in cloud infrastructure.

management plane and connectivity between the components through the partially restricted public network. Furthermore, additional tools may be useful to properly monitor distributed cloud infrastructure, for example, Prometheus[9], Telegraf[10] and Grafana[11]. Due to the increased number of virtual machines, cloud-based deployment benefits from the Packer tool automating also the creation of images.

Local testbed and cloud-based deployment do not differ significantly concerning the configuration files of 5G core network implementation. Once the infrastructure is properly configured (e.g. connectivity between regions and WireGuard tunnels) only IP addresses must be specified. However, there are significant, architectural differences between the environments. In the local testbed, all network functions are deployed within a single Kubernetes cluster. On the contrary, in a cloud-based deployment control and user planes as well as radio access network simulator are physically separated. It must not be the case, but we selected a fully distributed deployment architecture to present the advantages of cloud infrastructure. Finally, 5G network components are installed from the source files in the cloud environment, while in the local testbed Helm charts were used.

To sum up, the automation of cloud-based deployment is significantly more complex, requires additional steps, and comprises many more components, especially in the form of proposed distributed architecture.

6.5. Preliminary performance evaluation

We performed some preliminary performance evaluation of the proposed automation process for a cloud-based virtualized 5G network. To measure command execution time, the system command time was used. Ten measurements were performed for each test scenario to ensure statistical reliability. Based on the collected data, the maximum, minimum, average and median values were determined.

We measured the time needed to create the server infrastructure using Terraform modules, namely, the execution time of the terraform apply command. The measurement was performed for two scenarios:

- centralized (all network functions deployed in a single region of the

[9]https://prometheus.io/
[10]https://www.influxdata.com/time-series-platform/telegraf/
[11]https://grafana.com/oss/grafana/

cloud infrastructure),

- distributed (instances of UPF network function are distributed across several regions).

The results are presented in the Table 3.

Table 3: Environment creation time in minutes

	Minimum	**Maximum**	**Average**	**Median**
Centralized scenario	03:49	06:46	05:08	05:06
Distributed scenario	03:57	09:20	07:14	07:17

Terraform creates independent resources in parallel, so the command execution time was influenced by the time of the slowest resource created. The resource creation time depends on the performance of the cloud infrastructure. During testing, the resources that occasionally took the longest to create were virtual machines running in the *BHS5* region. It affects maximum, average and median times for distributed scenario. For example, the average creation time was two minutes longer for the distributed scenario, as in the centralized one no machine was deployed in the *BHS5* region. Simultaneously, when the performance of the whole cloud infrastructure was not deteriorated, both scenarios provided similar results (minimum environment creation time). To sum up, it is critical to properly select regions and be aware that the time needed to create the infrastructure may change depending on the performance of the cloud infrastructure.

We also measured the execution time of Ansible *playbook* considered as an execution time of the ansible−playbook command performed in the distributed scenario. We compared the times needed to deploy a virtualized core network with the use of virtual machines using:

- the base operating system (all the software must be installed during the deployment process),

- pre-installed images of virtual machines created with Packer (possible based on scripts proposed in Section 6.2).

The results are presented in Table 4.

The operations defined by a given *playbook* are performed sequentially. The average execution time difference was approximately ten minutes, as

Table 4: Execution time of the *ansible-playbook* command in minutes

	Minimum	**Maximum**	**Average**	**Median**
Base image	18:06	19:41	19:01	19:10
Pre-installed image	07:59	09:19	08:37	08:35

shown in Table 4. Consequently, minimum, average and median times are significantly different. It resulted from a smaller number of operations to be performed in the pre-installed case. In the case of the pre-installed image, all that was required was to configure the network and update the configuration of all tools. It shows the advantages of using pre-installed images of virtual machines as we proposed in this paper.

A significant reduction in the deployment time can be observed when using pre-installed virtual machine images. Such an optimization can be particularly important in the context of the automation of the deployment of virtualized 5G network. However, the use of pre-installed images also has its consequences, for example, it involves the use of a previously prepared version of the software, which may limit the flexibility of this solution.

7. Conclusions

This paper addresses the problem of automating deployment of virtualized mobile core for 5G and beyond mobile networks. 5G deployments will require automation to ensure robust, efficient and zero-touch deployment processes because of the increasing complexity and requirements on dynamic provisioning. Automation can be especially beneficial in the case of distributed deployments in complex cloud-based infrastructures. Typical use-cases are, for example, private mobile networks in Industry 4.0 or temporary networks for mass events to ensure effective transmission of video signals and reduce necessity of setting up wires.

The main goal of this paper was to describe a Proof-of-Concept testbed for automating the virtualized deployment of core mobile networks in a self-managed infrastructure, with additional insights about potential problems and solutions. The PoC includes, among others, technology stack, configuration files, images of virtual machines, modifications to the virtualization environment, configuration of kubernetes cluster, and comprehensive descriptions, facilitating reproducibility of the environment. All components are specific solely to 5G networks, and our solution is tailored specifically for

this application. Designing a framework specifically for the 5G core network allowed us to tackle challenges that are often neglected in current research. For example, we ensured interconnectivity between 5G components spread across multiple cloud environments and incorporated support for specialized protocols. Furthermore, the automation solution provides extensive configuration of the various 5G network elements.

Furthermore, we provided comprehensive tests and analysis for the automated deployments of Free5GC and Open5GS implementations in the 5G core network. The implementations were tested in terms of the throughput, the delay introduced by the UPF, and the time of UE registration. We compared two open-source core network implementations in terms of feasibility for virtualization and automation, an aspect overlooked in previous studies. We provided detailed descriptions explaining, for example, the preparation of the virtual environment, the process of creation and configuration of the Kubernetes cluster or Helm charts adoption to the underlying infrastructure. We also proposed modifications to Helm charts to remove the initial impediments in the automated 5G deployment process due to the required human intervention. All automation components were deployed and configured using Ansible playbooks, another original contribution of our work [5]. These playbooks create a Kubernetes cluster on the virtual machine, edit Helm charts, deploy the 5G core network using the charts, as well as, determine IP address of WEBUI service required to register a mobile subscriber and set up the UERANSIM subscriber.

We conduct a thorough performance evaluation of the automated 5G core network orchestration in a local environment (e.g. time needed to deploy core network) and based on the results, we propose enhancements that reduce deployment time by 40% on average for one of the implementations. We also present the process and necessary tools to automatically deploy a virtualized 5G core network in a cloud infrastructure. This contribution comprises, for example: creating and configuring cloud infrastructure, creating images of virtual machines, ensuring connectivity between distributed locations, and configuration of the virtualized 5G network. Similar fully automated cloud-based deployments are not currently available, even with the use of service orchestrators. The Open5GS open source implementation of the 5G core network was selected as a more prominent choice for cloud-based deployment. Finally, we performed preliminary performance assessments of the automation process in the proposed cloud-based deployment.

The contribution is expected to be useful for industry and academia in

experimenting with and developing automated orchestration systems for mobile core networks. Presented results may be especially important as requirements of consecutive generations of mobile networks become increasingly demanding. Meeting these requirements enables providing emerging services and satisfying end-users. The discussion of results may utilize architectural decisions and catalyze future research efforts.

Possible future research directions include providing a reliable comparison of open-source 5G core network implementations to commercial solutions. Additionally, we plan to extend the performance evaluation in the cloud-based deployment to better assess the automation process and also verify the performance of the geo-distributed user plane. Finally, to make our Proof-of-Concept more scalable to accommodate growing demands, a multi-node cluster with multiple UEs should be considered to host a virtualized mobile core network. Finally, the proposed automation framework can be further generalized, as it is currently tightly coupled to the 5G network workload.

Acknowledgement

This work was supported by the Polish Ministry of Science and Higher Education with the subvention funds of the Faculty of Computer Science, Electronics and Telecommunications of AGH University of Science.

References

[1] S. Zhang, An Overview of Network Slicing for 5G, IEEE Wireless Communications 26 (3) (2019) 111–117. doi:10.1109/MWC.2019.1800234.

[2] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, M. Zorzi, Toward 6G Networks: Use Cases and Technologies, IEEE Communications Magazine 58 (3) (2020) 55–61.

[3] Azure Private 5G Core, `https://azure.microsoft.com/en-us/products/private-5g-core`, description of charts directory - Helm (2024).

[4] R. Ullah, M. A. U. Rehman, M. A. Naeem, B.-S. Kim, S. Mastorakis, ICN with edge for 5G: Exploiting in-network caching in ICN-based edge computing for 5G networks, Future Generation Computer Systems 111 (2020) 159–174.

[5] Ansible playbooks automating deployment of Free5GC and Open5GS, `https://github.com/wojt3ek140/5g_core_automation` (2023).

[6] J. Rischke, P. Sossalla, S. Itting, F. H. P. Fitzek, M. Reisslein, 5G Campus Networks: A First Measurement Study, IEEE Access 9 (2021) 121786–121803. doi:10.1109/ACCESS.2021.3108423.

[7] M. Chepkoech, N. Mombeshora, B. Malila, J. Mwangama, Evaluation of open-source mobile network software stacks: A guide to low-cost deployment of 5g testbeds, in: 2023 18th Wireless On-Demand Network Systems and Services Conference (WONS), 2023, pp. 56–63.

[8] A. Sahbafard, R. Schmidt, F. Kaltenberger, A. Springer, H.-P. Bernhard, On the performance of an indoor open-source 5g standalone deployment, in: 2023 IEEE Wireless Communications and Networking Conference (WCNC), 2023, pp. 1–6.

[9] C. Wei, A. Kak, N. Choi, T. Wood, 5gperf: Profiling open source 5g ran components under different architectural deployments, Association for Computing Machinery, New York, NY, USA, 2022, p. 43–49.

[10] F. J. De Souza Neto, E. Amatucci, N. A. Nassif, P. A. Marques Farias, Analysis for Comparison of Framework for 5G Core Implementation, in: 2021 International Conference on Information Science and Communications Technologies (ICISCT), 2021, pp. 1–5. doi:10.1109/ICISCT52966.2021.9670414.

[11] L. B. Silveira, H. C. de Resende, C. B. Both, J. M. Marquez-Barja, B. Silvestre, K. V. Cardoso, Tutorial on communication between access networks and the 5g core, Computer Networks 216 (2022) 109301.

[12] A. Gabilondo, Z. Fernandez, A. Martin, R. Viola, M. Zorrilla, P. Angueira, J. Montalban, 5G SA Multi-vendor Network Interoperability Assessment, in: 2021 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), 2021, pp. 1–6. doi:10.1109/BMSB53066.2021.9547167.

[13] G. Lando, L. A. F. Schierholt, M. P. Milesi, J. A. Wickboldt, Evaluating the performance of open source software implementations of the 5G network core, in: IEEE/IFIP Network Operations and Management Symposium (NOMS 2023), 2023, pp. 1–7.

[14] D. Lake, N. Wang, R. Tafazolli, L. Samuel, Softwarization of 5G Networks–Implications to Open Platforms and Standardizations, IEEE Access 9 (2021) 88902–88930. doi:10.1109/ACCESS.2021.3071649.

[15] F. A. Wiranata, W. Shalannanda, R. Mulyawan, T. Adiono, Automation of Virtualized 5G Infrastructure Using Mosaic 5G Operator over Kubernetes Supporting Network Slicing, in: 2020 14th International Conference on Telecommunication Systems, Services, and Applications (TSSA, 2020, pp. 1–5. doi:10.1109/TSSA51342.2020.9310895.

[16] 5G-all-in-one Helm, `https://github.com/my5G/5G-all-in-one-helm`, 5G-all-in-one Helm (2024).

[17] I. Sanchez-Navarro, A. Serrano Mamolar, Q. Wang, J. M. Alcaraz Calero, 5GTopoNet: Real-time topology discovery and management on 5G multi-tenant networks, Future Generation Computer Systems 114 (2021) 435–447.

[18] C.-D. Phung, N.-E.-H. Yellas, S. B. Ruba, S. Secci, An Open Dataset for Beyond-5G Data-driven Network Automation Experiments, in: 2022 1st International Conference on 6G Networking (6GNet), 2022, pp. 1–4.

[19] B. Chun, J. Ha, S. Oh, H. Cho, M. Jeong, Kubernetes Enhancement for 5G NFV Infrastructure, in: 2019 International Conference on Information and Communication Technology Convergence (ICTC), 2019, pp. 1327–1329. doi:10.1109/ICTC46691.2019.8939817.

[20] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, P. Bertin, Improving traffic forecasting for 5g core network scalability: A machine learning approach, IEEE Network 32 (6) (2018) 42–49.

[21] Mijumbi, Rashid and Serrat, Joan and Gorricho, Juan-Luis and Bouten, Niels and De Turck, Filip and Boutaba, Raouf, Network function virtualization: State-of-the-art and research challenges, IEEE Communications Surveys & Tutorials 18 (1) (2016) 236–262. doi:10.1109/COMST.2015.2477041.

[22] R. Viola, A. Martín, M. Zorrilla, J. Montalbán, P. Angueira, G.-M. Muntean, A Survey on Virtual Network Functions for Media Streaming: Solutions and Future Challenges, ACM Comput. Surv. 55 (11) (feb 2023). doi:10.1145/3567826.

[23] I. Alam, K. Sharif, F. Li, Z. Latif, M. M. Karim, S. Biswas, B. Nour, Y. Wang, A Survey of Network Virtualization Techniques for Internet of Things Using SDN and NFV, ACM Comput. Surv. 53 (2) (apr 2020). doi:10.1145/3379444.

[24] N. F. Saraiva de Sousa, D. A. Lachos Perez, R. V. Rosa, M. A. Santos, C. Esteve Rothenberg, Network Service Orchestration: A survey, Computer Communications 142-143 (2019) 69–94. doi:https://doi.org/10.1016/j.comcom.2019.04.008.

[25] NG-RAN Architecture, https://www.3gpp.org/news-events/2160-ng_ran_architecture (2022).

[26] D. Mroziński, M. Klinkowski, K. Walkowiak, Cost-aware du placement and flow routing in 5g packet xhaul networks, IEEE Access 11 (2023) 12710–12726. doi:10.1109/ACCESS.2023.3241751.

[27] R. Goścień, Traffic-aware service relocation in software-defined and intent-based elastic optical networks, Computer Networks 225 (2023) 109660.

[28] 3GPP, 3rd Generation Partnership Project; 5G; System architecture for the 5G System (5GS) (Relase 16), Tech. Rep. TS 23.501 version 16.6.0, 3GPP (2020).

[29] Github page of UERANSIM project, https://github.com/aligungr/UERANSIM (2022).

[30] Quickstart, https://open5gs.org/open5gs/docs/guide/01-quickstart/, introduction to Open5GS (2022).

[31] S. Rommer, P. Hedman, M. Olsson, L. Frid, S. Sultana, C. Mulligan, Chapter 15 - selected call flows, in: S. Rommer, P. Hedman, M. Olsson, L. Frid, S. Sultana, C. Mulligan (Eds.), 5G Core Networks, Academic Press, 2020, pp. 395–429. doi:https://doi.org/10.1016/B978-0-08-103009-7.00015-6.
URL https://www.sciencedirect.com/science/article/pii/B9780081030097000156

[32] ESXi Wikipedia page, https://en.wikipedia.org/wiki/VMware_ESXi, eSXi wikipedia page (2022).

[33] Open5GS Ubuntu support, https://open5gs.org/open5gs/docs/platform/01-debian-ubuntu/, open5GS Ubuntu support (2022).

[34] Free5GC Environmental testing, https://github.com/free5gc/free5gc/wiki/Environment, free5GC Environmental testing (2022).

[35] GTP5G kernel module, https://github.com/free5gc/gtp5g, gTP5G kernel module (2022).

[36] Towards5GS Free5GC Helm Charts, https://github.com/Orange-OpenSource/towards5gs-helm, towards5GS Free5GC Helm Charts (2022).

[37] Openverso Open5GS Helm Charts, https://github.com/Gradiant/openverso-charts, openverso Open5GS Helm Charts (2022).

[38] Flannel is a network fabric for containers, designed for Kubernetes, https://github.com/flannel-io/flannel, flannel (2023).

[39] Using the Multus CNI in OpenShift, https://cloud.redhat.com/blog/using-the-multus-cni-in-openshift, types of CNI plugins (2022).

[40] Kubernetes Pod Lifecycle, https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/, kubernetes Pod Lifecycle (2022).

[41] BusyBox: The Swiss Army Knife of Embedded Linux, https://busybox.net/about.html, details about busybox (2022).