A Brief Performance Comparison of Bare-Metal and Kubernetes Deployments for 5G Core Control Plane Network Functions using Open5GS

Tariro Mukute
Electrical Engineering
University of Cape Town
Cape Town, South Africa

Mathias Santos de Brito
Electrical Engineering
Technische Universität Berlin
Berlin, Germany

Albert A. Lysko
CSIR NGEI
Council for Scientific and Industrial Research
Pretoria, South Africa

Joyce Mwangama
Electrical Engineering
University of Cape Town
Cape Town, South Africa

Thomas Magedanz
Electrical Engineering
Technische Universität Berlin
Berlin, Germany

***Abstract*—This paper investigates the performance difference of critical 5G User Equipment (UE) procedures when deployed on a Kubernetes platform versus a traditional bare-metal deployment. We leverage Open5GS, an open-source implementation of the 5G Core (5GC), to evaluate the impact of containerisation on key performance metrics. The research answers (i) how the performance of critical 5G UE procedures differs when 5GC is deployed on a Kubernetes environment compared to a traditional bare-metal deployment and (ii) provides the measurable cost introduced by Kubernetes in terms of key 5G performance metrics. We evaluated throughput and latency. The paper analyses the observed performance differences against theoretical expectations arising from the Kubernetes architecture overhead and insights from related work. Our study reveals a 7% performance degradation in throughput for UE procedures running on Kubernetes compared to bare-metal when handling more than 300 initiated UE devices.**

***Index Terms*—5G Core, Kubernetes, Load Testing, Network Traffic Generator, NFV, Open5GS, Testbed**

I. Introduction

The fifth generation (5G) of mobile communication technology promises a significant leap forward in data speeds, network capacity, and responsiveness. This transformation is driven by the core network, which is responsible for critical tasks such as user authentication, session management, and data routing. Traditionally, these functions were implemented as monolithic software applications on dedicated hardware. However, the demands of 5G necessitate a more agile and scalable approach.

To meet these demands, the concept of Service Based Architecture (SBA) for 5G was standardised by the Third Generation Partnership Project (3GPP) in Release 15. This approach decomposes the core network into smaller, independent Network Functions (NFs) that can be deployed and managed more efficiently [1]. Container orchestration technologies like Kubernetes, Docker Swarm, and Apache Mesos have emerged as powerful tools for managing these NFs. They help achieve Agility, Scalability, Resource Efficiency, and Vendor Independence [2], [3]. Kubernetes is considered a de-facto candidate for the orchestration of 5G and beyond in a cloud environment, as it has the largest community, better support, and studies suggest that its usage and adoption will increase in the future [4].

Open5GS is an open-source implementation of the 3GPP 5th Generation Core Network (5GC) standards. Its microservice architecture makes it ideal for containerisation and deployment on Kubernetes. This alignment between Open5GS and Kubernetes enables the exploitation of containerisation advantages within 5G Core network deployments.

The deployment of 5GC Network Function (NF)s on containerisation technologies like Kubernetes is gaining traction due to its potential benefits [4]. However, understanding the performance implications of such deployments is crucial, especially for control plane procedures critical for real-time service delivery in 5G networks. Several studies have explored various performance aspects related to 5GC on Kubernetes, but there are limitations that our work aims to address. Existing research often focusses on specific areas such as network communication frameworks [5], orchestration efficiency [6] [7], containerised Network Function Virtualization (NFV) deployments [8], Container Network Interface (CNI) plugin performance [9] [10], or user experience [4]. Although some studies evaluate performance metrics for 5GC, they often concentrate on the Data Plane (User Plane) or orchestration aspects, neglecting the Control Plane performance of 5GC NFs on Kubernetes [6], [7], [11]

Our work addresses this gap by investigating the performance of essential 5G Control Plane procedures (Registration, and De-registration) under various numbers of User equipment (UE)s on these procedures on Open5GS deployed on Kubernetes. We measure key performance indicators (KPIs), latency and throughput, to assess the suitability of this approach

for real-world 5G deployments. This focus on Control Plane performance and the use of Open5GS sets our research apart from existing work. By investigating these critical procedures, we aim to provide a more comprehensive understanding of the impact of containerisation on the core functionality of 5G.

Our contributions include:

1) Evaluating the impact of Kubernetes on the Control Plane performance of Open5GS.
2) Providing insights into the real-world suitability of containerised 5G Core deployments.
3) Developing tools and scripts to automate deployments and performance monitoring.

The remainder of this paper is structured as follows. Section 2 provides background information on 5G Core and Kubernetes. Section 3 explores related research on 5G Core deployments on Kubernetes. Section 4 details the methodology of our research, including the experimental setup, workload generation, and performance measurement techniques. Section 5 presents the results of our experiments, analysing the performance of Open5GS on Kubernetes compared to a bare-metal deployment and discusses the implications of our findings. Finally, Section 7 concludes the paper by summarising our key contributions and highlighting potential future research directions.

II. Background

This section provides an overview of the 5G core network architecture and its key procedures, followed by a brief explanation of Kubernetes and its functionalities relevant to 5G core deployment. We will conclude by discussing the advantages of deploying 5G core on Kubernetes and the potential overhead introduced by Kubernetes compared to bare-metal deployments.

The 3GPP standardised the 5GC architecture starting in Release 15. This architecture defines NFs such as the Access Management Function (AMF), Session Management Function (SMF), and User Plane Function (UPF) [1]. These NFs work together to handle essential functionalities such as user registration, session management, and data traffic forwarding. 5GC Procedures, are a series of operations performed by NFs, they are crucial for achieving 5GC functionalities. These include registration procedure, Protocol Data Unit (PDU) session establishment procedure, and handover procedure, among others. The time it takes to complete these procedures directly impacts the delay experienced by user applications, highlighting the importance of control plane performance. The different 5GC procedures and operations are detailed in 3GPP Technical Specification (TS) [12]–[15].

In this study, we leverage Open5GS, an open-source implementation of the 5GC NFs developed in C and using a MongoDB database for state storage. Other alternatives 5GC implementations are Open5GCore, OpenAirInterface 5GC and Free5GC. Our choice of Open5GS is based on our previous comparative study [16] in which Open5GS demonstrated the highest performance.

Kubernetes is an orchestration framework that facilitates containerisation, orchestration, and scaling of applications. It utilises a master-worker architecture, where the master node controls the worker nodes that run the containerised applications [17] (in our case, the 5GC NFs). Key components of the worker nodes include:

- *Kubelet*: Manages Pods (ensuring they're running and healthy) and interacts with the container runtime to start and stop containers.
- *Container Runtime*: Manages container lifecycles (e.g., Docker, containerd).
- *Kube-proxy*: Manages network connectivity for Pods and services.
- *Add-ons*: Optional functionalities like CoreDNS for service discovery and network plugins (CNI) for container networking.

There are several advantages to deploying 5G core on Kubernetes [3], these include:

- *Agility*: Faster deployment and updates of network functions.
- *Scalability*: Easy scaling of network functions based on traffic demands.
- *Resource Efficiency*: More efficient resource utilisation through containerisation.
- *Self-healing*: Automatic recovery from failures for improved service availability.
- Vendor Independence: Enables deployment of network functions from different vendors.

Although Kubernetes offers several benefits, it also introduces some overhead compared to bare-metal deployments. These overheads can stem from:

- *Networking*: CNI plugins can introduce overhead compared to native networking on bare metal. The specific overhead depends on the chosen CNI plugin, and the workload. [18] [5] [19].
- *Process Management*: Kubernetes manages container lifecycles through the kubelet and container runtime, adding overhead compared to directly running applications on bare metal [20].
- *Memory*: Containerisation adds additional memory overhead due to container images and runtime environments [20].

This study quantifies the performance impact of this overhead on 5GC control plane procedures deployed on Kubernetes compared to a bare-metal baseline. The study provides insights into the practical impact of Kubernetes overhead on 5GC control plane network performance.

III. Related Works

Kubernetes offers potential benefits in agility, scalability, and manageability of 5GC deployments [4]. Several studies have explored various performance aspects related to 5GC on Kubernetes, but there are limitations that our work aims to address. Existing research often focuses on specific areas such as network communication frameworks, orchestration

efficiency, containerised NFV deployments, CNI plugin performance, or user experience. Although some studies evaluate performance metrics, they often concentrate on the Data Plane (User Plane) or orchestration aspects, neglecting the Control Plane performance of 5GC NFs on Kubernetes.

For example, Osmani et al. [5] compared network communication frameworks for 5GC workloads. They found Network Service Mesh (NSM) to perform on par with Calico for intra-cluster communication, while machine-to-machine (M2M) communication exhibited the best performance. However, their study did not consider the impact on 5GC NFs processing capabilities, which could potentially limit the overall performance gains. Similarly, Arouk et al. [4] compared the deployment and user experience performance of the bare-metal, Docker, and Kube5G-Operator setups. Although Kube5G-Operator offered faster deployment times, the network performance (TCP and UDP traffic) showed negligible differences between the three deployments. This suggests that the evaluation focused on the User Plane (Data Plane), and the Control Plane performance of 5GC NFs was not analysed. Tufeanu et al. [11] also face a similar limitation in their work evaluating the Open Air Interface (OAI) on Kubernetes. Their work validated the control plane functionality but used iperf for data transfer performance, which primarily reflects Data Plane performance rather than Control Plane metrics on Kubernetes.

Studies on container orchestration performance provide valuable insights on the overhead of Kubernetes deployments but do not directly address our research focus. Khichane et al. [6] compared the efficiency of NFV orchestration using a cloud-native approach (CNF-Orchestrator on Kubernetes) and a Virtual Machine (VM)-based approach. They reported significant improvements in deployment and upgrade times with the cloud-native approach. Luong et al. [7] investigated orchestration metrics in Kubernetes and Marathon for 5G deployments. Their focus was on container and service initialisation times, not including performance metrics of the 5GC NFs themselves. They also analysed CPU usage by increasing the number of user equipment (UE). Ungureanu et al. [8] evaluated inter-process communication (IPC) between containerised Open5GS applications using APIs for cloud and edge deployments. They observed better response times in the cloud setup compared to the edge, along with higher memory consumption in the edge control plane deployment.

The performance of the CNI plug-in is another relevant area of research. Budigiri et al. [9] investigated the performance overhead of security network policies on Kubernetes. They compared Calico and Cilium eBPF with the host network mode. Although eBPF offered advantages, the overhead of CNI plugins remained significant, especially for inter-node communication. They also observed a higher overhead on OpenStack compared to bare-metal deployments. Qi et al. [10] conducted a qualitative and quantitative comparison of open-source CNI plugins. Their analysis reveals that CNI plugins add performance overhead, and the overhead is different for each CNI provider. They also highlight that the overhead increases as the workload increases.

In conclusion, existing research provides valuable insight into various aspects of 5GC deployments on Kubernetes. However, a gap exists in comprehensively evaluating the Control Plane performance of 5GC NFs deployed on Kubernetes, as summarised in Table I. Our work aims to address this gap by investigating the performance of essential 5GC Control Plane procedures, namely Registration and De-registration. We measure the key performance indicators (KPIs), latency, and requests throughput, to assess the suitability of this approach for real-world 5G deployments.

TABLE I: Summary of Related Work

Author	**Area**	**Focus**
Osmani et al. [5]	Network Performance	NFs vs Network communication
Arouk et al. [4]	User Experience	Deployment, User Experience
Tufeanu et al. [11]	OAI on Kubernetes	Control Plane Functionality, Data Plane Metrics
Khichane et al. [6]	Orchestration Performance	Cloud vs. VM Orchestration
Luong et al. [7]	Orchestration Performance	Initialization Times, CPU Usage
Ungureanu et al. [8]	Container Orchestration	IPC between Open5GS
Budigiri et al. [9]	CNI Plugin Performance	Security Policy Overhead
Qi et al. [10]	CNI Plugin Performance	Open-source CNI Plugins

IV. Methodology

This section details the experimental setup, workload generation methodology, and metrics used to evaluate the performance of 5G core procedures on Kubernetes and bare-metal deployments. The goal is to minimise bias and ensure reproducibility. We will establish two environments: (i) *Bare-Metal Deployment*: This represents the baseline for comparison; we provide the Ansible playbooks to reproduce the deployment [21] and (ii) *Kubernetes Environment*: This environment will host the containerised 5G core network functions, we provide Helm charts to reproduce the environment [22]. We make use of 6 servers, on which we install Open5GS directly for the bare-metal deployment, and install Kubernetes (microk8s flavour version 1.29.4) for the Kubernetes deployment. The servers run a version of Ubuntu. Table II describes the hardware and software specifications for the setup.

A. 5G Core Procedures and Participating NFs

This section details the 5G Core procedures chosen for performance evaluation and the Network Functions (NFs) involved in each. We will measure key performance indicators (KPIs) similar to the approach used in Goshi et al. [23]. We focus on the Registration and De-registration procedures. The flow of the participating NFs' procedures is shown in Figure 1. The UE initiates a registration request with the UE security parameters to the AMF (Step 1), which performs the UE authentication using information stored in the Unified

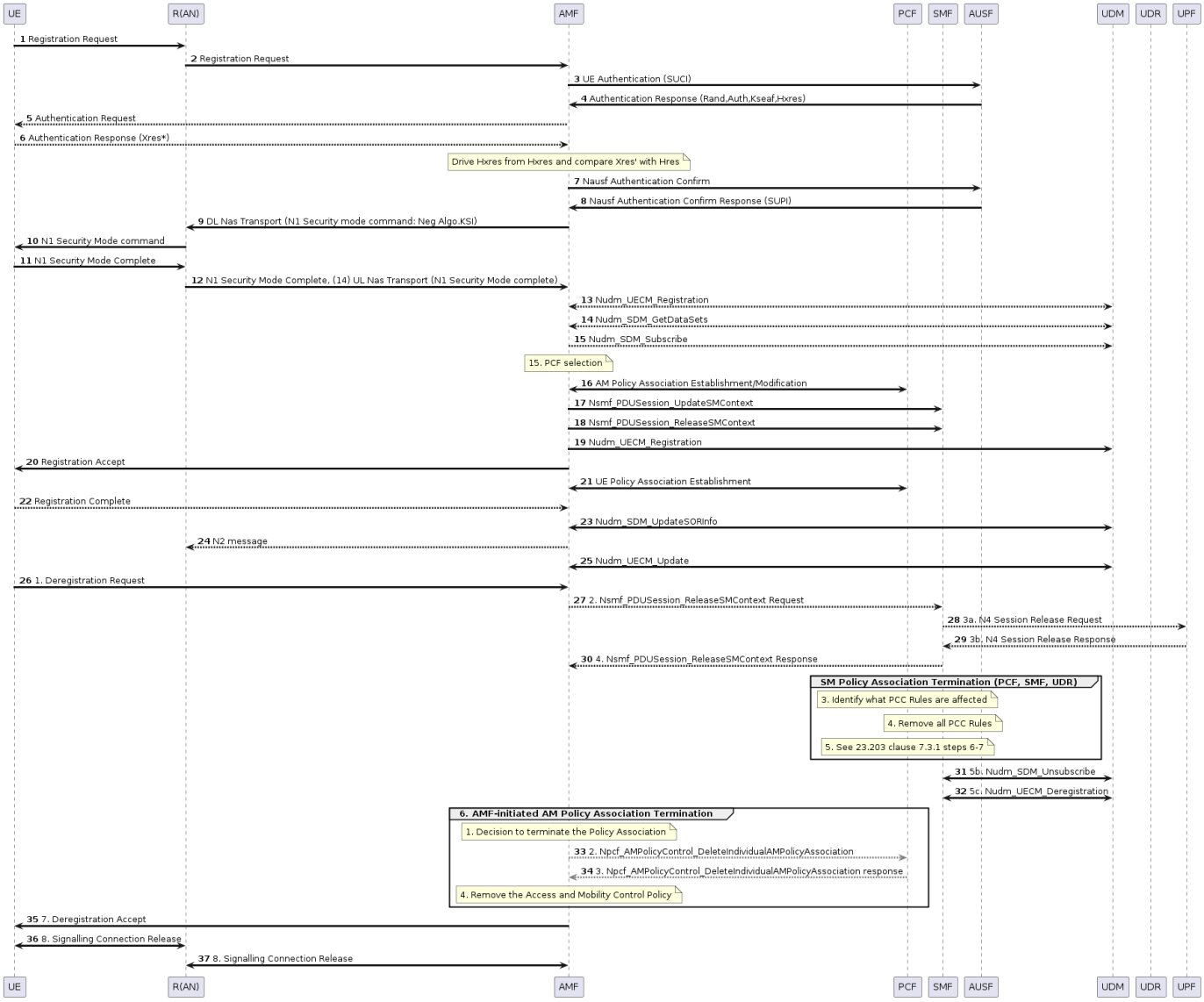


Fig. 1: Registration + De-Registration Procedure

Data Management (UDM) (Steps 3 - 8). Upon successful authentication, the AMF sends a security mode command to the UE to establish the security parameters for the connection and updates relevant databases, including the UDM, User Data Repository (UDR), and SMF (Steps 9 - 19). The AMF then sends a registration accept message to the UE, to which the UE optionally responds with a ”registration complete” response. For reregistration, the UE initiates a deregistration request, and the AMF updates the relevant databases and releases the signalling connection (Steps 20 - 37). The AMF plays a crucial role in coordinating these processes and ensuring seamless communication between the UE and the network. Open5GS includes other network functions not included in Figure 1, namely, Binding Support Function (BSF), Service Communication Proxy (SCP), Network Slice Selection Function (NSSF) and Network Repository Function (NRF)

Our performance evaluation centres on latency and throughput measurements, considering the entire lifecycle of a UE's registration and deregistration. We utilise a specialised traffic generator [24] to initiate the Registration Request for each UE. Once a UE completes the Registration procedure (indicated by the Registration Complete Non-Access Stratum (NAS) message), the traffic generator will immediately trigger the Deregistration procedure for that specific UE.

To gather these measurements, the traffic generator captures the procedures' operations/requests' start and end timestamps for each UE. For Registration, we will measure the time between the Registration Request and Registration Complete NAS messages. Additionally, we will capture the timestamps for individual NAS messages exchanged between Registration Request and Registration Accept, namely, Authorisation Request/Response (Steps 5 - 6 in Figure 1) and Security Mode

TABLE II: Open6GNet Testbed Cluster Nodes

	Hardware			Software		
	CPU	Memory	Kubernetes Role	OS	Kernel-Version	NF Placement
node1	i7-6700T (4x Cores)	8GB	control-plane	Ubuntu 22.04.4 LTS	5.15.0-105-generic	
node2	i7-6700T (4x Cores)	32GB	worker	Ubuntu 22.04.3 LTS	5.15.0-91-generic	UDR & SCP & NSSF
node3	i7-7700T (4x Cores)	32GB	worker	Ubuntu 23.04	6.2.0-39-generic	BSF & NRF
node4	i7-6700T (4x Cores)	16GB	worker	Ubuntu 22.04.3 LTS	5.15.0-106-generic	UDM & PCF
node5	i7-6700T (4x Cores)	32GB	worker	Ubuntu 22.04.3 LTS	5.15.0-88-generic	AMF & AUSF & MongoDB & WebUI
node6	i5-6700T (2x Cores)	8GB	worker	Ubuntu 22.04.3 LTS	5.15.0-88-generic	SMF & UPF

Command/Complete (Steps 10 - 11 in Figure 1) messages. For Deregistration, we collect the timestamps for the Deregistration Request NAS message, Deregistration Accept NAS message (Steps 26 - 35 in Figure 1) and the UE Context Release Complete NG Application Protocol (NGAP) message. We collect the results of three iterations and present the averages with two key metrics:

1) **Latency - UE Round-Trip Time (Registration to Deregistration):** This captures the total time a UE takes, from sending the Registration Request to completing Deregistration (including UE Context Release). We present the average round-trip time as the number of UEs increases, comparing performance on Kubernetes and bare-metal deployments. This allows us to assess the impact of containerisation using Open5GS.
2) **Throughput - Number of requests serviced per window period:** To understand the relationship between procedure execution time and throughput of UEs, we calculate the number of UE requests concurrently served by the 5GC core at defined intervals (window periods). On each deployment, we show this for 500 UEs at 0.1-second intervals. This visualises the number of UEs being handled by the 5GC at defined intervals and the rate at which the UE operations are served.

By combining these latency and throughput measurements, we comprehensively evaluate the performance of the chosen 5G Core procedures when deployed on a Kubernetes platform versus bare-metal using Open5GS.

V. Results and Discussion

This section presents the key findings of the performance evaluation of the 5GC core procedures deployed on a containerised platform using Open5GS. We focus on two primary metrics, *Latency* - UE Round-Trip Time and *Throughput* - Number of requests serviced per window period , comparing a Kubernetes deployment with a bare-metal setup. We employed a traffic generator [24] that initially sent registration requests at a rate of approximately 20 UEs per 0.1 seconds.

A. Latency - UE Round-Trip Time (Registration to De-registration)

This metric represents the average time it takes for a User Equipment (UE) to complete the entire process, starting from Registration request to De-registration completion. Our results, depicted in Figure 2a show that for a low to moderate number of UEs (up to around 200), both deployments exhibit similar round-trip times with a fairly linear increase as the number of concurrent UEs is increased. However, beyond 200 UEs, the bare-metal deployment demonstrates a lower average round-trip time, suggesting that it can handle increasing load more efficiently.

B. Throughput - Number of Requests Serviced per Window Period

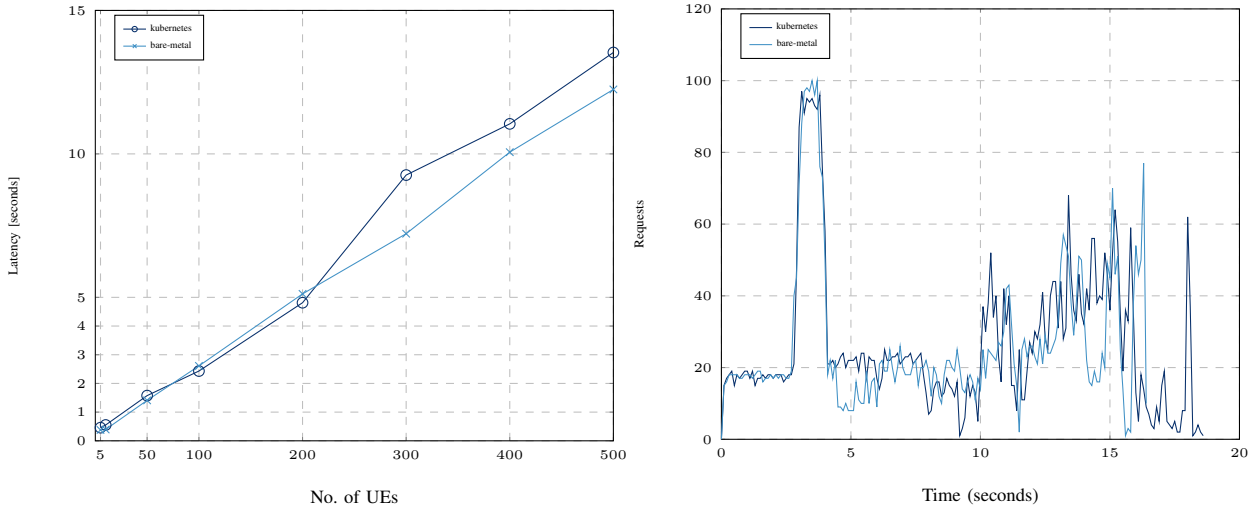
The throughput reflects the number of requests processed by the system within a defined time window (0.1 seconds in this case). Throughput was measured by running the traffic generator until a steady state with 500 UEs was reached on both deployments. Figure 2b shows the throughput of the requests per window period. The results indicate that the bare-metal deployment services a peak of 100 requests per 0.1-second window, while the Kubernetes deployment reaches a maximum of 97 requests per window at the same UE count. On the majority of the plot, Kubernetes has relatively higher requests per window, compared to bare-metal. It's important to not that higher throughput, or request per second, can lead to depletion of system resource, resulting in latency and jitter. Figure 2b additionally shows that the bare-metal completes the requests earlier than the kubernetes deployment. This corresponds to the maximum time taken to complete the UE Registration and De-Registration procedures highlighted in Figure 2a.

The observed performance difference between bare-metal and Kubernetes deployments can be attributed to several factors discussed in the Background and Related Works sections. These include containerisation overhead, resource management strategies, the CNI used, among other factors. For instance, Qi et al [18], observed that the CNI plugin performance overhead increase with increasing workload. This can additionally explain the degrading performance for Kubernetes deployment depicted in Figure 2a.

Although both deployments achieved successful registration and de-registration procedures for a moderate number of UEs, the bare-metal setup demonstrates a 7% performance advantage at 300 UEs. This highlights a potential trade-off between flexibility and raw performance when considering containerised deployments for 5G Core networks.

VI. Conclusion and Future Work

This study evaluated the performance of essential 5G Core Control Plane procedures deployed on a containerised platform utilising Open5GS. A comparative analysis was conducted



(a) Average time taken (by UE) to Complete 5G Procedures (UE Registration + De-Registration Proc)

(b) Requests per given window period (0.1 seconds). This is the sum of the different requests.

Fig. 2: Performance comparison of Open5GS on bare-metal vs Kubernetes deployment

between a Kubernetes deployment and a bare-metal setup. The findings indicate that, under moderate loads of less than 200 UEs with registration requests sent at a rate of approximately 20 UEs per 0.1 seconds, both deployments exhibit comparable performance. However, under increased workload, over 300 UEs, the bare-metal deployment exhibited lower latency and higher throughput under increasing workloads. These results underscore the necessity of considering the trade-off between flexibility and raw performance when deploying 5GC core NFs on containerised platforms. Although Kubernetes provides benefits in terms of manageability and portability, it introduces a 7% performance overhead, which can impact performance and stability at elevated loads. The decision to prioritise Kubernetes deployment over a bare-metal setup may be influenced by the relative operational costs (OPEX) associated with each. We leave this exploration to the discretion of the readers and 5GC operators.

Future research can explore optimisation techniques within the Kubernetes environment to address the observed performance differences. This could involve profiling the computing resource usage and processes, investigating containerisation overhead mitigation strategies, resource management improvements, and exploring more robust configurations to enhance stability under high load. Additionally, evaluating the performance of different CNI plugins and container orchestration platforms might provide valuable insights for optimising containerised 5GC deployments.

Furthermore, extending the evaluation to encompass a wider range of 5GC core procedures and investigating the performance under real-world traffic scenarios with varying request arrival rates would provide a more comprehensive understanding of the suitability of containerised deployments for 5G Core networks.

REFERENCES

[1] *3GPP TS 23.501*, ETSI, 2018, [Online; accessed 21-June-2023]. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138400_138499/138401/16.03.00_60/ts_138401v160300p.pdf

[2] F. A. Wiranata, W. Shalannanda, R. Mulyawan, and 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*. IEEE, 2020, pp. 1–5.

[3] H. Mfula, A. Ylä-Jääski, and J. K. Nurminen, "Seamless kubernetes cluster management in multi-cloud and edge 5g applications," in *International Conference on High Performance Computing & Simulation*, 2021.

[4] O. Arouk and N. Nikaein, "Kube5g: A cloud-native 5g service platform," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[5] L. Osmani, T. Kauppinen, M. Komu, and S. Tarkoma, "Multi-cloud connectivity for kubernetes in 5g networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 42–47, 2021.

[6] A. Khichane, I. Fajjari, N. Aitsaadi, and M. Gueroui, "Cloud native 5g: an efficient orchestration of cloud native 5g system," in *NOMS 2022-2022 IEEE/IFIP network operations and management symposium*. IEEE, 2022, pp. 1–9.

[7] D.-H. Luong, H.-T. Thieu, A. Outtagarts, and Y. Ghamri-Doudane, "Cloudification and autoscaling orchestration for container-based mobile networks toward 5g: Experimentation, challenges and perspectives," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*. IEEE, 2018, pp. 1–7.

[8] O.-M. Ungureanu, C. Vlădeanu, and R. Kooij, "Collaborative cloud-edge: A declarative api orchestration model for the nextgen 5g core," in *2021 IEEE international conference on service-oriented system engineering (SOSE)*. IEEE, 2021, pp. 124–133.

[9] G. Budigiri, C. Baumann, J. T. Mühlberg, E. Truyen, and W. Joosen, "Network policies in kubernetes: Performance evaluation and security analysis," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2021, pp. 407–412.

[10] S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Understanding container network interface plugins: design considerations and performance," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*. IEEE, 2020, pp. 1–6.

[11] L.-M. Tufeanu, A. Martian, M.-C. Vochin, C.-L. Paraschiv, and F. Y. Li, "Building an open source containerized 5g sa network through docker and kubernetes," in *2022 25th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. IEEE, 2022, pp. 381–386.

[12] 3GPP, "TS 123 502 - v17.8.0 - 5G; Procedures for the 5G System (5GS) (3GPP TS 23.502 version 17.8.0 Release 17)," ETSI, Tech. Rep., 2023. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/17.08.00_60/ts_123502v170800p.pdf

[13] 3GPP, "TS 123 527 - V17.6.0 - 5G; 5G System; Restoration procedures (3GPP TS 23.527 version 17.6.0 Release 17)," ETSI, Tech. Rep., 2021. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123527/17.06.00_60/ts_123527v170600p.pdf

[14] 3GPP, "TS 132 290 - V17.6.0 - 5G; Telecommunication management; Charging management; 5G system; Services, operations and procedures of charging using Service Based Interface (SBI) (3GPP TS 32.290 version 17.6.0 Release 17)," ETSI, Tech. Rep., 2021. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/132200_132299/132290/17.06.00_60/ts_132290v170600p.pdf

[15] 3GPP, "TS 133 501 - V15.16.0 - 5G; Security architecture and procedures for 5G System (3GPP TS 33.501 version 15.16.0 Release 15)," ETSI, Tech. Rep., 2021. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.16.00_60/ts_133501v151600p.pdf

[16] T. Mukute, L. Mamushiane, A. A. Lysko, R. Modroiu, T. Magedanz, and J. Mwangama, "Control plane performance benchmarking and feature analysis of popular open-source 5g core networks: Openairinterface, open5gs, and free5gc," *IEEE Access*, pp. 1–1, 2024.

[17] S. K. Mondal, R. Pan, H. D. Kabir, T. Tian, and H.-N. Dai, "Kubernetes in it administration and serverless computing: An empirical study and research challenges," *The Journal of Supercomputing*, pp. 1–51, 2022.

[18] S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Assessing container network interface plugins: Functionality, performance, and scalability," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 656–671, 2020.

[19] G. Koukis, S. Skaperas, I. A. Kapetanidou, L. Mamatas, and V. Tsaoussidis, "Performance evaluation of kubernetes networking approaches across constraint edge environments," *arXiv preprint arXiv:2401.07674*, 2024.

[20] L. Espe, A. Jindal, V. Podolskiy, and M. Gerndt, "Performance evaluation of container runtimes." in *CLOSER*, 2020, pp. 273–281.

[21] T. Mukute, "Performance benchmarking of 5G Core networks," 2024. [Online]. Available: https://tariromukute.github.io/control_plane_performance_analysis

[22] M. S. De Brito, E.-R. Modroiu, B. Q. Le, T. Magedanz, M. Corici, and J. Mwangama, "OPEN6GNET: A Learning and Experimentation Platform Based on Open-Source Solutions and Cloud-Native Approaches," pp. 1–6, 2024.

[23] E. Goshi, R. Stahl, H. Harkous, M. He, R. Pries, and W. Kellerer, "Pp5gs-an efficient procedure-based and stateless architecture for next generation core networks," *IEEE Transactions on Network and Service Management*, 2022.

[24] T. Mukute, "5G Core Network Traffic Generator," 2023. [Online]. Available: https://github.com/tariromukute/CoreNetworkTrafficGenerator