Procedure-Aware Stateless Systems for 5G & Beyond Core Networks

Endri Goshi†, Vignesh Karunakaran†, Hasanin Harkous*, Rastin Pries*, Wolfgang Kellerer†
†Technical University of Munich, *Nokia
E-Mail:†{endri.goshi, vignesh.karunakaran, wolfgang.kellerer}@tum.de,
*{hasanin.harkous, rastin.pries}@nokia.com,

***Abstract*—As public and private cloud-native deployments of the 5G Core (5GC) networks are rolling-out on a wide scale, attention is shifting towards efficient state management. While stateful deployments were the default method in the previous generations of mobile networks, they lack the necessary flexibility that cloud-native orchestration demands. Yet, traditional approaches taken to enable stateless deployments require the operators to sacrifice on performance due to the frequent state transactions. To overcome this issue, in this paper we propose a Piggyback-based and a Proactive-Push approach which allow for procedure-aware stateless 5GC systems. Our evaluations highlight the advantages of the Piggyback approach for two synchronous control procedures, reducing their completion time by $\sim$44% and $\sim$70% compared to the baseline. For asynchronous procedures, the Proactive-Push approach outperforms the baseline with $\sim$13% and $\sim$22%. More importantly, these mechanisms do not pose additional overhead on CPU and bandwidth utilization.**

I. Introduction

The introduction of the 5th Generation (5G) of mobile communication networks has drastically changed the way how the future core networks (CN) will be designed and operated. The advances in Software-Defined Networking (SDN) and Network Function Virtualization (NFV) paved the way for the adoption of these paradigms in the telecommunication domain. The concept of Control and User Plane Separation (CUPS) was initially introduced by 3GPP in Release 14 [1] to increase the network management flexibility and enhance CN's performance when handling an increased volume of traffic, thus laying the foundations of the novel 5G Core (5GC) design. Guided by the requirements of new 5G-powered use-cases for more scalable, flexible, and agile CN architectures, the Service-Based Architecture (SBA) was introduced in Release 15 [2].

The 5G SBA adopts cloud-native design principles such as the modularization of the Network Functions (NF) and a client-server model for the inter-NF communication via the Service-Based Interfaces (SBIs). The high functional decomposition resulted in a range of new NFs, e.g.,: Access and Mobility Function (AMF), Authentication Server Function (AUSF), Network Repository Function (NRF), Network Slice Selection Function (NSSF), Policy Control Function (PCF), Session Management Function (SMF), Unified Data Management (UDM) and Unified Data Repository (UDR).

To fully leverage the advantages of cloud-native environments, such deployments are characterized by the *stateless* nature of the NFs. Unlike traditional *stateful* applications, a stateless NF implements a logical compute-storage separation by maintaining the state information in a remote database (DB), thus, allowing for flexible and efficient orchestration. A stateless 5GC architecture is noted by the deployment of the Unstructured Data Storage Function (UDSF) as the state management DB. For the SBA system shown in Fig. 1, AMF, AUSF, SMF, PCF and UDM will store their User Equipment (UE) context in this DB.

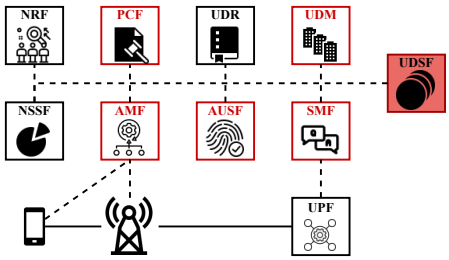

Fig. 1. Stateless 5G Core Architecture. NFs in red boxes maintain their state in UDSF. NRF and UDR are stateless by design, while NSSF does not maintain any UE-specific information.

Due to frequent state transactions, stateless deployments may come with a high cost in terms of increased control plane latency and hindered system's scalability [3], [4]. In this work, we demonstrate that these performance concerns can be addressed by designing procedure-aware stateless systems which reduce the frequency of state transactions, while still providing orchestration flexibility. To this end, we propose and develop two approaches, namely Piggyback-based and Proactive-Push state management mechanisms. These systems are deployed in a private cloud-native environment and compared with a baseline stateless 5GC. Our extensive evaluations show that the completion time of control procedures in stateless 5GC can indeed be improved, while not causing additional overhead in terms of CPU and bandwidth utilization. We believe that the contributions of this work are beneficial to the research community and for the deployment of cloud-native 5G & Beyond CNs as the performance penalty for building resilient and flexible NFs can be alleviated.

The remainder of the paper is structured as follows. In Section II, we provide background information regarding the stateful and stateless paradigms. A summary of related work

is presented in Section III. In Section IV we introduce the procedure-aware systems and provide details on their implementation, while the evaluation setup and results are presented in Section V. Lastly, Section VI summarizes future directions and concludes the paper.

II. Background

In cloud-native environments, the NFs are commonly deployed as containerized applications. This procedure represents a lightweight virtualization technique that provides adequate levels of isolation without the large runtime overhead of using Virtual Machines (VMs) [5]. The management of these containerized workloads and services is done through container orchestration tools such as Kubernetes (K8s) [6] that try to efficiently utilize the available resources. However, cloud environments are highly dynamic and containers are frequently moved between the machines or horizontally scaled to adapt to the incoming traffic. Moreover, containers are not designed as highly available entities, therefore making container failures a common occurrence.

Due to the reasons mentioned above, state management techniques play an important role in the deployment strategies for mobile core networks. In general, we can distinguish between *stateful* and *stateless* NFs.

Stateful NFs maintain locally the entire context that is necessary for them to operate correctly. This allows them to start processing requests as soon as they arrive, as illustrated by `NF-3` in Fig. 2. However, the critical drawback of stateful NFs is that the state, and consequently the processing of a request, is bound to a specific instance. If the orchestrator decides to terminate the running instance or an internal failure occurs, the state information is lost and thus directly affecting the services.

Stateless NFs on the other hand, implement a processing logic and state management separation. In this case, the state is not maintained locally anymore, but stored in a remote state DB which is queried every time the NF needs access to this information. Likewise, the updated state information is sent to the DB after the NF has finished processing the request. This separation is important for containerized workloads as it helps the system recover quickly from unexpected failures. In addition, it simplifies the orchestration process in scenarios where a scale-out operation is necessary to handle the incoming traffic (i.e., the number of instances is increased). However, the flexibility of stateless NFs comes with the cost of: i) increased processing time, and ii) communication overhead.

To facilitate the deployment of stateless 5GC, 3GPP introduced the Unstructured Data Storage Function (UDSF) to serve as the state DB [7]. Statelessness in 5GC can be implemented at a *transactional* or *procedural* level [3]. A transaction represents a single interaction between two NFs (e.g., request/response), while a procedure can comprise a set of transactions. To better understand the difference, we refer to Fig. 2 where `NF-2` depicts a transactional stateless application. The start of the transaction is marked by the incoming request. Before starting to process it, `NF-2` needs to query UDSF for the UE context information. Likewise, as the transaction finishes, the updated context is sent to UDSF.

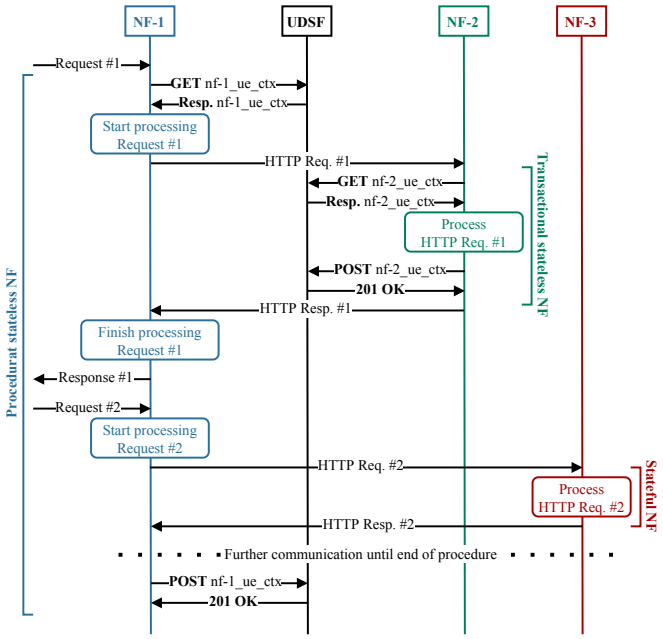

Fig. 2. A hybrid system where the NFs implement three different types of state management approaches: i) *procedural* stateless shown in blue, ii) *transactional* stateless shown in green, and iii) *stateful* shown in red.

Considering the high number of transactions between NFs, this approach imposes high overhead. `NF-1` on the other hand, implements a procedural stateless approach and only queries and sends the UE context at the beginning and the end of a procedure. Compared to the transactional approach, it is less granular as the state is maintained locally for the duration of the procedure. Hence, the main difference lies in the fact that a transactional stateless NF can recover from failures that occur during a procedure execution, with the cost of higher bandwidth utilization and control procedure completion times.

III. Related Work

In the recent years, many works have researched the advantages and the impact that stateless deployments have in the context of mobile core networks. A stateless Mobility Management Entity (MME) was introduced in dMME [8], allowing for geographically distributed deployments with remote storage. Developed as a highly available and distributed architecture, ECHO [9] maintains state in an external entity which ensures reliability using an end-to-end distributed state machine replication protocol. An alternative implementation to stateless MME was proposed in MMLite [10]. It is designed as a stateless and fully decomposed MME, where each control procedure is implemented in its own microservice. In [11], three techniques are proposed to reduce latency due to state management in 4G: i) bypassing sequential flows, ii) pipelining control data, and iii) parallelizing control procedures. However, these systems focus only on the legacy 4G core networks.

ML-SLD [12] introduces a message-level stateless design for cloud-native 5GC, based on the NAS/NGAP communication protocols. They propose the servitization of the RAN-Core

interface through a middleware called RAN Integrated Service Enabler. In [3], the authors evaluate the impact of stateless AMF, SMF and UPF in 5GC SBA. They propose mechanisms leveraging state-sharing among NFs and exploit parallel execution to reduce the cost of transactional statelessness. A piggybacking mechanism in proposed in [13], similar to the one in this work. However, their approach considers only AMF, SMF and UPF, unlike our Piggyback-based approach that spans all the involved NFs in the execution of control procedures. Also, we present and evaluate a Proactive-Push approach for efficient state management. Lastly, the impact on performance of transactional stateless 5GC is highlighted in [4], where it is shown that such deployments scale poorly due to the frequent state-related communication. The procedure-aware systems proposed in this paper are evaluated and compared against the Stateless Free5GC that is introduced in [4].

IV. Procedure-Aware State Management

For UEs to be able to register with the operator and communicate with other UEs or exchange data with services in the Internet, a set of control procedures need to be executed. These procedures represent the sequences of communication that must take place in 5GC in order to manage the life-cycle of the UE, as introduced in [14]. In this work, we consider four important control plane procedures: i) *Registration* attaches the UE to 5GC, ii) *PDU Session Establishment* establishes the data plane communication, iii) *PDU Session Release* terminates the data plane session, and iv) *Deregistration* detaches the UE from the network.

In principle, procedural stateless NFs provide enough orchestration flexibility and with a lower cost than transactional stateless NFs. However, most of the 5GC NFs (with the exception of AMF and SMF) are not procedure-aware (i.e., they cannot deduce which procedure is being executed). Thus, leaving transactional statelessness as the only feasible approach to be implemented [4]. To tackle this issue, in this work we introduce two approaches that aim to bring procedure-awareness to the 5G & Beyond CNs. The *Piggyback-based* and *Proactive-Push* state management techniques are the result of a thorough analysis based on the 3GPP specifications and empirical data gathered from the deployment of Stateless Free5GC. In the following subsections, we provide details regarding the design and implementation of these two approaches. In both cases, we leverage AMF's procedure-awareness to achieve our goal.

A. Piggyback-based State Management

In computer communications, *piggybacking* refers to the process of appending additional data in a request/response in order to reduce the bandwidth utilization of the network. In a stateless 5GC, a piggyback-based state management approach helps by making the state information available to NFs at the same time that they receive the request to be processed.

First, `global_ue_ctx` is defined as a data structure that maintains the UE context of all the stateless NFs, on a per UE basis. The `global_ue_ctx` is essentially a dictionary that contains entries with the name of the NF as key and their specific UE context information as data. Moreover, parts

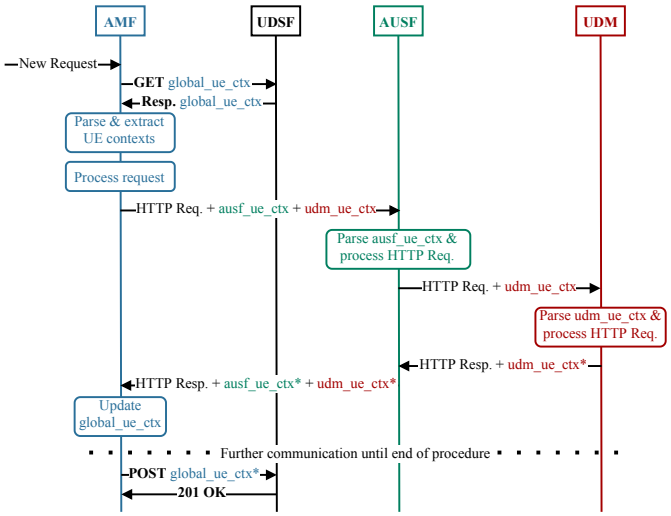

Fig. 3. Communication diagram of a 5G system implementing the *Piggyback-based* approach.

of UE context previously kept in many NFs simultaneously are included only once in this structure, avoiding redundancy and saving resources. In the Piggyback-based system that we propose, the `global_ue_ctx` is queried from UDSF only at the beginning of the procedure and the updated version is then stored at UDSF at the end of the procedure. Being procedure-aware and the entry point of control-plane traffic to 5GC, we have selected AMF as the only NF that performs these tasks.

To better understand the communication flow, we refer to Fig. 3. When AMF receives a NAS/NGAP message that marks the start of a new control procedure (e.g., Initial UE Registration Request), it requests the latest version of `global_ue_ctx` from UDSF. Next, AMF parses this information into its local memory and uses the `amf_ue_ctx` to process the request it received. As the communication progresses, AMF sends HTTP requests to other control plane NFs, e.g., `UE_Authentication_Request` sent to AUSF. If AUSF is the only NF that will be involved in this operation, AMF will append `ausf_ue_ctx` to the `UE_Authentication_Request` as a piggyback. If that is not the case, and e.g., AUSF will send a `Generate_Auth_Data` request to UDM, AMF will also piggyback the `udm_ue_ctx` to the same request that it sent to AUSF. While it would be easier to forward the entire `global_ue_ctx` between NFs, a decision was made to forward only the relevant information because it is more efficient in terms of network bandwidth utilization. Similar to how the contexts are initially propagated via requests, the other NFs piggyback the updated information to the HTTP response messages that they send (denoted with * in Fig. 3). Once they reach AMF, the information in `global_ue_ctx` is updated accordingly.

To implement this state management approach, the following steps are taken:

- The procedural call-flows are analyzed to identify the NF chains that are triggered after every request that AMF

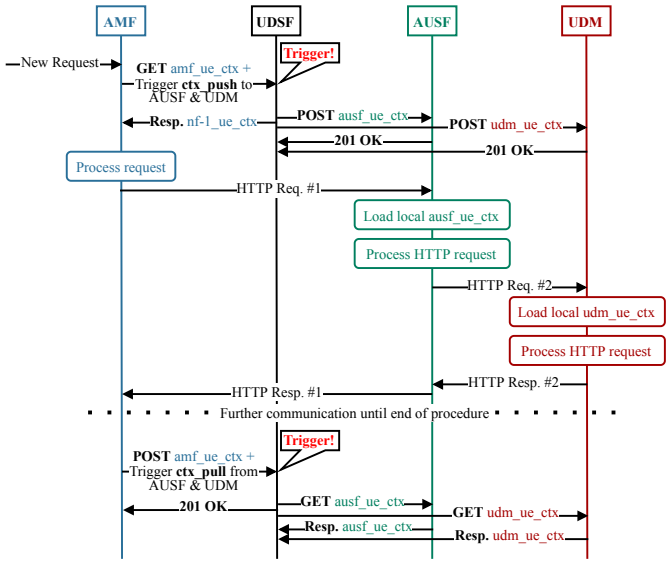

Fig. 4. Communication diagram of a 5G system implementing the *Proactive-Push* approach.

sends. Based on this analysis, AMF's implementation is modified to enable piggybacking the UE contexts.

- The UE state machine in AMF is leveraged to mark the start and finish of control procedures. The processing logic is modified such that AMF can communicate with UDSF and (de)serialize the `global_ue_ctx`.
- Other Stateless Free5GC NFs' implementation is extended to: i) parse context from incoming requests and if applicable append it to the next-in-line NF, and ii) the set of SBIs that NFs expose now includes endpoints that are able to process piggybacked information in addition to the HTTP request. To simplify the operation, all the NFs are made aware of structure of the `global_ue_ctx` and know how to parse it, but AMF is the only responsible NF to update its information. Modifications are thus performed on AUSF, SMF, PCF, and UDM.

B. Proactive-Push State Management

Aiming at making the UE context available to the NFs before they receive requests, we design and implement a Proactive-Push state management technique. In essence, the system that we propose differs from the default pull-based approach of Stateless Free5GC in that the UE context is pushed from UDSF to the control plane NFs, and not the other way around. Contrary to the piggyback-based approach, this technique does not require defining a data structure like the `global_ue_ctx`. Instead, we maintain the same UE context definitions as in Stateless Free5GC [4].

Referring to Fig. 4, we can observe that after receiving a request from the UE, marking the start of a control procedure, AMF contacts UDSF to get the latest version of `amf_ue_ctx`. However, in this proposed system, this request is modified such that it includes trigger information. In addition to its own UE context, AMF asks UDSF to push UE context information to the other involved NFs, which will maintain a

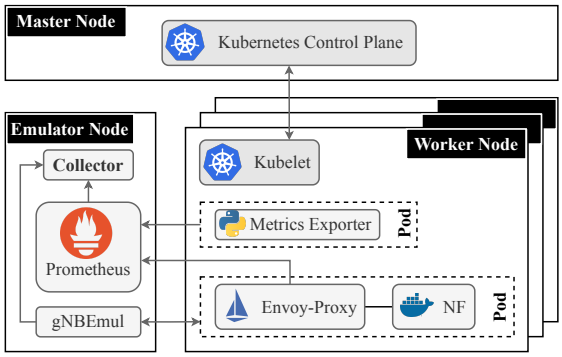

Fig. 5. Overview of the K8s testbed setup for the evaluations.

copy of this information in a local dictionary until the end of the procedure. From this point on, each NF in the chain will be able to quickly access the UE context from the local dictionary. When AMF determines that the procedure has finished, it will store its `amf_ue_ctx` and at the same time trigger a *pull* operation on the UDSF. In other words, UDSF will now query the latest UE context information from each NF, as specified by AMF in the request. Once this operation finishes, each NF deletes their local copy of UE context.

From an implementation perspective, this approach requires the following:

- AMF's implementation is augmented with information on the involved 5GC NFs in each control procedure.
- UDSF API endpoints are extended to be able to understand trigger information sent from AMF.
- The set of SBIs that AUSF, SMF, PCF and UDM expose now include new endpoints, enabling them to: i) receive state information via POST requests and store it locally, and ii) receive GET requests and respond with their latest UE context information.

V. Performance Evaluation

To assess and compare the performance improvements of the procedure-aware stateless systems, we consider a private cloud-native environment. To this end, we set up a K8s-orchestrated cluster consisting of 1 *master* and 11 *worker* nodes (machines), shown in Fig. 5. To avoid cases where 5GC NFs compete for the same physical resources, we distribute their deployment to separate nodes (i.e., one NF per node). A homogeneous set of DELL OptiPlex 9020 workstations is used for worker machines, each equipped with 16 GB of RAM and an octa-core Intel i7-4770 CPU running at 3.40 GHz. Moreover, by interconnecting the nodes using a 1Gb switch, we create an isolated environment and avoid network interference. In addition to 5GC, in the K8s cluster we deploy an evaluation framework, consisting of:

- *Prometheus* [15] – a monitoring application with widespread use in K8s environments. Its main function in our testbed is to collect metrics during measurements and make them available for post-processing.
- *Envoy-Proxy* [16] – deployed as a sidecar container, it intercepts all the traffic to and from an NF. While its

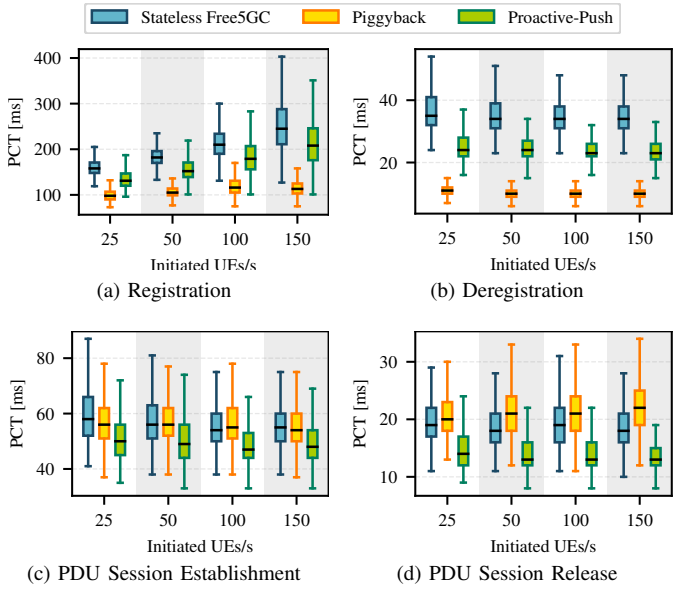

Fig. 6. Procedure Completion Times w.r.t. the number of new initiated procedures per second for each of the considered control procedures. Outliers are omitted for better visualization.

primary functionality is to facilitate traffic routing, it also collects HTTP metrics which are later queried from Prometheus.

- *Metrics Exporter* – a Python application deployed in each worker node, collecting metrics every 1 s and reporting them to Prometheus. The main metrics collected are the CPU and memory utilization values.
- *gNBEmu* – a gNB & UE Emulator able to generate high volume of 5G control plane traffic (UE requests). It achieves this by emulating control procedures' communication for an arbitrary number of UEs, sending real requests and responses to 5GC. Each procedure is timestamped to calculate their completion time.

With respect to the Key Performance Indicators (KPIs), this paper focuses on the average CPU utilization of each system, and Procedure Completion Times (PCTs) for four control plane procedures: i) Registration, ii) PDU Session Establishment, iii) PDU Session Release, and iv) Deregistration. Other KPIs such as memory utilization and total payload of the HTTP communication are evaluated but not presented in this paper due to space limitations. Nonetheless, the results show that the performance of the procedure-aware systems on these two KPIs is very similar to the baseline Stateless Free5GC system. In the baseline, AMF is procedural stateless while the rest of the NFs are transactional stateless. The number of new UEs/s, which represents the scale of the input traffic, is varied between $[25, 50, 100, 150]$, with new arrivals uniformly distributed with an interarrival time of 3 ms. Each measurement runs for 45 s and 8 campaigns executed for each scenario. The following subsections present and discuss the results of our evaluations.

A. Procedure Completion Times

Procedure Completion Time (PCT) is defined as the time it takes for the execution of a control procedure to finish, measured from the moment the first message is sent from the UE, until the last response or acknowledgment is received from 5GC. With the goal of comparing the impact of different stateless approaches in control plane latency, PCT represents the main KPI in our evaluations.

During the execution of the Registration procedure (see Fig. 6a), piggybacking the UE contexts leads to a completion time reduction of $\sim$44% compared to the baseline, while the Proactive-Push approach achieves an improvement of $\sim$14%. Moreover, by observing the trend for the different input traffic, we see that the Piggyback approach shows a considerably better performance in terms of scalability, compared to both the baseline and Proactive-Push. During Deregistration, the situation is similar to the Registration procedure, as shown in Fig. 6b. The Piggyback-based system performs the best with an improvement of $\sim$70% compared to the baseline. The Proactive-Push approach on the other hand, achieves an improvement of $\sim$30%.

However, the situation changes for the next two procedures, as shown in Fig. 6c and Fig. 6d. The Piggyback approach exhibits a very similar performance to the baseline in the case of PDU Session Establishment, but an increased average PCT of $\sim$17% during PDU Session Release. The reason behind this behavior is the asynchronous nature of communication during these procedures, which nullifies the benefits of piggybacking the state information. Therefore, the overhead that comes from processing the `global_ue_ctx` leads to the increased PCTs. Nonetheless, because of its design, the Proactive-Push approach is not affected by the asynchronous communication, while still reaping the benefits of proactively making the state information available to the NFs. Thus, the Proactive-Push approach achieves lower PCTs than the other systems in both these procedures. It completes the PDU Session Establishment and PDU Session Release $\sim$13% and $\sim$22% faster, respectively, compared to the baseline.

B. CPU Utilization

Resource utilization is an important metric to evaluate the efficiency of cloud-native 5GC deployments as it can be directly linked to the system's scalability. Hence, we provide a comparison of the CPU utilization of the proposed procedure-aware systems and the baseline Stateless Free5GC. In Fig. 7, the average CPU utilization values of 5GC NFs are shown, collected over 8 independent measurement campaigns. In each bar plot, we distinguish between: i) AMF shown in a hatched pattern, ii) UDSF shown in gray, and iii) the rest of the involved NFs shown in solid colors. NFs' average utilization values are stacked and the height of the bars represents the total system CPU utilization.

Compared to the Stateless Free5GC, the Piggyback approach reduces the overall CPU utilization during the Registration and Deregistrations procedures, by $\sim$10% and $\sim$38%, as shown in Fig. 7b and Fig. 7d. Comparing only AMF, we observe that its utilization is in fact slightly higher than the baseline. This happens because in the Piggyback approach AMF has to (de)serialize the entire `global_ue_ctx` instead of only its own UE context, and process HTTP packets with

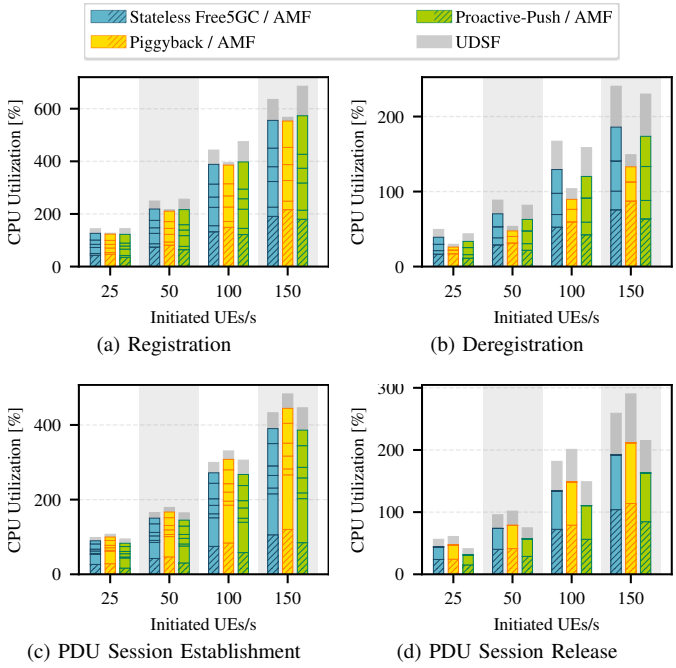

Fig. 7. Average CPU utilization of 5GC NFs w.r.t. the number of initiated procedures per second for each of the considered control procedures.

larger payload (due to the piggybacked state information). This is compensated with the decreased utilization of UDSF, since it is only contacted by AMF at the beginning and the end of the procedure. These values hold true for all the configurations of input traffic. However, during PDU Session Establishment and PDU Session Release, this approach consumes $\sim$10% more resources. In the case of PDU Session Establishment shown in Fig. 7c, this is due to an increased CPU utilization from AMF and SMF, again stemming from processing the `global_ue_ctx`. In the case of PDU Session Release shown in Fig. 7d, the overall increase is attributed to AMF only, for the same reasons as the other procedures.

The Proactive-Push approach on the other hand, outperforms the other systems only during PDU Session Release where it utilizes $\sim$21% less CPU resources, on average. During Deregistration it performs only slightly better than the baseline, while during PDU Session Establishment it exhibits similar resource consumption. Proactive-Push shows the highest utilization relative to the other systems during the Registration procedure, where it shows $\sim$8% more CPU utilization in the scenario with 150 UEs.

The results of our evaluations suggest that there is not a single optimal procedure-aware system that exhibits the best performance for all the control procedures. The involvement of different NFs in the control procedures means that the communication patterns and the complexity of their interworkings play a big role. As such, the Piggyback approach achieves the best performance in terms of both PCT and CPU utilization during procedures that are completely synchronous. While the Proactive-Push approach outperforms the baseline during all the procedures in terms of PCT, compared to the Piggyback approach it performs better only during procedures with asynchronous communication patterns. Therefore, a hybrid procedure-aware implementation would be able to achieve optimal performance. A detailed analysis of the communication patterns for the most critical procedures would allow for a correct classification of them into synchronous and asynchronous. Then, based on this classification, a hybrid system can be implemented by following a converged approach that uses the `global_ue_ctx` and the push-triggers in UDSF.

VI. Conclusion

With the deployment of cloud-native 5GC systems reaching a wider scale, attention is shifting towards efficient stateless implementations. Traditional transactional stateless approaches in the control plane suffer from increased latency and resource utilization. In this work, we introduce two procedure-aware approaches that aim to overcome the impact of statelessness on control plane latency. The measurements conducted in our testbed confirm the advantages of the Piggyback-based and Proactive-Push approaches during synchronous and asynchronous procedure execution, respectively. Most importantly, these come at no additional CPU and bandwidth utilization costs compared to the transactional stateless baseline. In the future, we plan to develop a hybrid system leveraging both these approaches, hence achieving optimal performance regardless of the interworkings of the control procedures.

References

[1] 3GPP, "3GPP TS 23.002 - Network architecture (Rel 14)," 2017.

[2] ——, "3GPP TS 23.501 - System architecture for the 5G System (Rel 15)," 2021.

[3] U. Kulkarni, A. Sheoran, and S. Fahmy, "The cost of stateless network functions in 5G," in *Proc. of ACM ANCS*, 2021.

[4] 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 TNSM*, 2022.

[5] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A comparative study of containers and virtual machines in big data environment," in *Proc. of IEEE CLOUD*, 2018.

[6] "Kubernetes," https://kubernetes.io/, [Accessed April-2023].

[7] 3GPP, "TS 29.598 - Unstructured data storage services (Rel 16)," 2020.

[8] X. An, F. Pianese, I. Widjaja, and U. G. Acer, "dMME: Virtualizing LTE mobility management," in *Proc. of IEEE LCN*, 2011.

[9] B. Nguyen, T. Zhang, B. Radunovic, R. Stutsman, T. Karagiannis, J. Kocur, and J. Van der Merwe, "ECHO: A reliable distributed cellular core network for hyper-scale public clouds," in *Proc. of ACM MobiCom*, 2018.

[10] V. Nagendra, A. Bhattacharya, A. Gandhi, and S. R. Das, "MMLite: A scalable and resource efficient control plane for next generation cellular packet core," in *Proc. of ACM SOSR*, 2019.

[11] Y. Li, Z. Yuan, and C. Peng, "A control-plane perspective on reducing data access latency in LTE networks," in *Proc. of ACM MobiCom*, 2017.

[12] K. Du, L. Wang, X. Wen, Y. Liu, H. Niu, and S. Huang, "ML-SLD: A message-level stateless design for cloud-native 5G core network," *Digital Communications and Networks*, 2022.

[13] U. Kulkarni, A. Sheoran, and S. Fahmy, "Towards a low-cost stateless 5G core," in *Proc. of IEEE LANMAN*, 2022.

[14] 3GPP, "3GPP TS 23.502 - Procedures for the 5G System (5GS)," 2022.

[15] Prometheus Authors, "Prometheus," https://prometheus.io/, [Accessed April-2023].

[16] Envoy Project Authors, "Envoy Proxy," https://www.envoyproxy.io/, [Accessed April-2023].