PP5GS—An Efficient Procedure-Based and Stateless Architecture for Next-Generation Core Networks

Endri Goshi, *Graduate Student Member, IEEE*, Raffael Stahl, Hasanin Harkous, Mu He, *Member, IEEE*, Rastin Pries, and Wolfgang Kellerer, *Senior Member, IEEE*

***Abstract*—The introduction of the Service-Based Architecture (SBA) for the 5G Core Networks has drastically changed the way these networks are designed and operated. Aiming for higher flexibility and agility, the adoption of SBA is the first step towards cloud-native deployments of 5G Core. However, the high degree of functional decomposition in SBA has implications in terms of increased inter-NF signaling traffic during the execution of control plane procedures, as well as an increased complexity in orchestrating a system with tight inter-NF dependencies. In this work, we introduce PP5GS as a stateless 5G Core architecture that implements a procedure-based functional decomposition of the 5G Core NFs. We develop Per-Procedure NFs for four different control plane procedures and perform extensive evaluations in a private cloud environment orchestrated with Kubernetes. The results show that PP5GS requires up to 34% and 55% less computing resources compared to the baseline stateful and stateless systems, respectively, while generating at least 40% less signaling traffic. Moreover, complex control plane procedures can complete up to 50% faster. Lastly, the results show that PP5GS is a more feasible architecture in leveraging edge-offloading of 5G Core NFs.**

***Index Terms*—5G core, SBA, control plane, performance measurements.**

I. Introduction

THE 5TH Generation (5G) of mobile communication networks is envisioned to support a wide range of new use-cases and an ever-increasing number of connected devices. The high-bandwidth requirements of enhanced Mobile BroadBand (eMBB) necessitate smaller cell sizes, and consequently increase mobility-related traffic [1], [2]. Moreover, massive Machine-Type Communication (mMTC) introduces the large-scale deployment of IoT devices and other end-user equipment with high control to data traffic ratio [3]. Therefore, from a control plane perspective the exploding number of connected devices and the unprecedented volume of control traffic lead to concerns of scalability and potential signaling storms [4].

Guided mainly by the flexibility limitations of the previous generations and the lack of agility in developing new features, 3rd Generation Partnership Project (3GPP) introduced the Service-Based Architecture (SBA) for the 5G Mobile Core Network (5GC) in its Release 15 [5]. In the recent years, Internet applications tend to incorporate cloud-native design principles to tackle their flexibility and agility issues. Therefore, the adoption of SBA aims to introduce cloud-native concepts such as microservices to 5GC deployments. SBA follows an approach of a high degree of functional decomposition coupled with a granular task distribution among the Network Functions (NFs). While this simplifies each component from an individual perspective, it does the opposite from a system perspective. The increased number of NFs involved in serving incoming traffic as well as the number of messages exchanged between them have lead to very high signaling overhead [6]. Consequently, the increased signaling overhead has shown to have a direct impact on the fulfillment of signaling traffic Service-Level Agreements (SLAs) [7].

Furthermore, as users are usually concerned about the experienced data plane latency, the untimely processing of control plane traffic can have a direct effect on the data plane access latency [8]. SBA inherently produces tight inter-NF dependencies which may cause scalability issues and increased control plane latencies if not tackled by complex orchestration mechanisms [9]. In the light of this, efficiently leveraging distributed cloud environments for 5GC deployments becomes a difficult task.

The main research question that we address in this work is whether we can perform a different functional split for the 5G & Beyond Core Networks that aims to increase the performance of mobile networks while still maintaining a high degree of flexibility, necessary for deployments in cloud environments. While recent research works have introduced improved designs [10], [11] and protocol simplifications [6], [12] for legacy 4G Core Networks, this work aims to solve practical issues with the current SBA deployments.

In order to develop a feasible architecture for the current and the Next Generation Mobile Networks, the main challenge lies on designing a system able to solve the following

Manuscript received 31 August 2022; revised 9 December 2022; accepted 14 December 2022. Date of publication 19 December 2022; date of current version 9 October 2023. The associate editor coordinating the review of this article and approving it for publication was A. Santos. *(Corresponding author: Endri Goshi.)*

Endri Goshi, Raffael Stahl, and Wolfgang Kellerer are with the Chair of Communication Networks, Technical University of Munich, 80333 Munich, Germany (e-mail: endri.goshi@tum.de; raffael.stahl@tum.de; wolfgang.kellerer@tum.de).

Hasanin Harkous and Rastin Pries are with the Nokia Standards Department, Nokia, 81541 Munich, Germany (e-mail: hasanin.harkous@nokia.com; rastin.pries@nokia.com).

Mu He was with the Nokia Standards Department, Nokia, 81541 Munich, Germany. He is now with NIO, 81679 Munich, Germany (e-mail: mu.he@tum.de).

Digital Object Identifier 10.1109/TNSM.2022.3230206

problems: i) high inter-NF signaling overhead observed during the execution of control plane procedures; ii) increased latencies or procedure completion times; and iii) *constrained* NF placement in a distributed 5GC infrastructure. Therefore, to address performance concerns of the distributed SBA, in this work we propose a novel architecture for 5G & Beyond that implements a procedure-based functional split for the control plane NFs. The Per-Procedure and Stateless 5G (PP5GS) architecture achieves this by means of consolidating most of the procedure processing logic in a single Per-Procedure NF (PPNF). The diversity of the number and type of the involved NFs in different procedures means that each PPNF can process only the corresponding procedure, e.g., a *RegNF* serves the incoming UE Registration traffic. The consolidation of the processing logic greatly reduces the generated traffic overhead and eliminates some of the impacts of network conditions in the experienced procedure completion time. Moreover, PPNFs are simpler to orchestrate because there are less inter-NF dependencies to be considered and the system's performance can be better estimated. In addition, PP5GS offers function-level scalability in contrast to the instance-level scalability in 5GC SBA. PP5GS is shown to be more resource efficient and that is an important factor in edge deployments where the resources are scarce. In addition, we identify an issue that is usually overlooked with respect to the edge-offloading of 5G SBA NFs. In its current state, offloading control plane NFs to the edge can be counter-productive as the cost of inter-NF communication for procedure execution can lead to degraded performance. By design, PPNFs can overcome this issue since the processing logic is almost entirely integrated into a single NF. Further, we argue that PP5GS brings added value in use-cases where knowledge of incoming control traffic can be leveraged to accelerate critical procedures, e.g., in an airport scenario with frequent Registration and PDU Session Establishment procedures the operator can accelerate their completion by deploying the relevant PPNF closer to the users. Nonetheless, the advantages apply to a centralized 5GC deployment as well. In [13], we have discussed overall aspects of the motivation and high level design of a 5GC system that implements a procedure-based functional decomposition. However, the system presented there serves only as a proof of concept with only a partial implementation and evaluation of *RegNF*. This paper makes the following contributions:

- We develop a stateless 5GC by implementing a 3GPP-compliant Unstructured Data Storage Function (UDSF) and enabling the other NFs to store their state remotely and retrieve it when needed.
- We implement a gNodeB (gNB) & User Equipment (UE) emulator that can generate control traffic in a scalable way for a range of procedures. Additionally, we develop a User Plane Function (UPF) emulator that can handle the high input rates of PDU Session related procedures.
- We design PP5GS as a stateless and procedure-based system for 5GC. Four PPNFs are implemented to handle the execution of Registration, PDU Session Establishment, PDU Session Release and Deregistration procedures.
- We deploy PP5GS in a Kubernetes cluster and perform detailed evaluations, comparing it against the baseline stateful 5GC as well as the stateless version. PP5GS outperforms the other systems in terms of CPU utilization, procedure completion time and communication overhead. Additionally, we highlight its superiority in edge offloading scenarios.

The remainder of the paper is structured as follows. Section II presents related work in the areas of mobile core network designs and stateless systems. Background information with respect to 5GC and technologies used in this work is given in Section III. PP5GS system's design and implementation are explained in Section IV, while in Section V we present the evaluation setup and the obtained results. Lastly, Section VI provides a discussion on the results and concludes the paper.

II. Related Work

Re-architecting the mobile core network: CleanG [6] leverages Network Function Virtualization (NFV) and Software Defined Networking (SDN) to design an architecture that minimizes the control plane latency as well as the data plane updates latency. In their architecture, the authors propose a single control plane NF and a single data plane NF with consolidated logic, while considerably simplifying the control plane protocols. TurboEPC [12] considers the Evolved Packet Core (EPC) of 4G and leverages data plane programmability to offload some control plane tasks to the data plane. By modifying the division between control and data plane tasks, a significant fraction of signaling traffic is offloaded. SoftBox [10] and PEPC [11] propose similar *EPC-in-the-box* solutions. In Softbox, both the control and data plane processing logic is consolidated in the same container, on a per-UE basis. On the other hand, PEPC only consolidates the UE state in EPC in a single location, while efficient access to the state is achieved by reorganizing the NFs. In Neutrino [14], the authors redesign the Mobility Management Entity (MME) by introducing a consistency protocol capable of ensuring fast failure recovery, a fast serialization engine and proactive geo-replication. The adoption of technologies such as NFV, SDN or data plane programmability are now a reality in 5G. In this work, we argue that while softwarization brings great flexibility to 5GC, there are also issues that need to be tackled. Our system aims to explore better ways to leverage the flexibility of softwarization, and at the same time minimize the ramifications on performance.

Similar to our approach, authors in [15] propose to improve the performance of Long Term Evolution (LTE) during the execution of critical events by means of "logic-based NF segregation". However, their solution's scope is limited to an offloading mechanism deployed alongside EPC, unlike PP5GS where our goal is to have a fully functioning 5G and Beyond Core with a per-procedure functional split. MobileStream [16] decomposes the functional blocks of monolithic EPC entities, and leverages real-time stream processing frameworks to assemble them into control plane pipelines. While this framework offers scalability and programmability, it is still

a distributed architecture where the stream communication between the blocks can span between several machines and thus it can suffer from similar issues as SBA. On the other hand, PP5GS is still distributed from a system point of view, but the centralization of the processing logic on a per-procedure basis into the PPNFs makes it possible to tackle the overhead of the distributed architectures. Recently, the work in [17] has introduced an architecture where the core network is conceptualized as a single large-scale and distributed Web service, with the functional split following the system procedures and services instead of the network functions. While they argue about the benefits of their architecture, they have not performed any evaluations. PP5GS is deployed in a private cloud environment and thoroughly evaluated for varying procedures and deployment scenarios.

Lastly, to the best of our knowledge, there has been no previous work that investigates the problems with offloading 5GC control plane NFs to the edge. In this work, we identify a critical issue that stems from the adoption of SBA in cloud environments that span to the edge, and detail how our system can overcome it.

Stateless mobile core: MMLite [18] and dMME [19] implement and investigate the scalability of stateless MME entities. More specifically, MMLite presents a stateless and fully decomposed MME entity where each control procedure is implemented in its own microservice. dMME proposes the deployment of geographically distributed MMEs with remote storage. SCALE [4] does not implement statelessness but instead proposes a two-tier architecture for MMEs with load-balancers and processing entities. It uses state replication to synchronize the deployed instances. In ECHO [20], a distributed and highly available EPC architecture tailored for public cloud deployments is presented. State is stored in an external entity and reliability is ensured with an end-to-end distributed state machine replication protocol. Lastly, authors in [21] investigate the impact of introducing statelessness to the 5G SBA. Their focus is on designing mechanisms that reduce the cost of transactional statelessness by sharing the user state among NFs and exploiting parallelism in execution. However, they only focus on Access and Mobility Function (AMF), Session Management Function (SMF) and UPF.

In this work we enable statelessness for all the 5G SBA NFs, contrary to state of the art works which focus either on a set of 5GC NFs or on EPC entities. Such stateless architecture is adequate for cloud deployments where state updates occur frequently. Furthermore, we introduce PP5GS as a procedural stateless architecture with high performance gains compared to stateless SBA.

III. Background

In this section, we look into the architecture of 5G Core Networks (5GC) with its most important features and how the communication between its entities is realized in the control plane. Moreover, we provide details on a 5GC project that is considered as the baseline for our work. The major changes in designing and operating 5GC are empowered by the introduction of the cloud-native paradigm in this domain. Therefore, we highlight Kubernetes [22] as a integral technology and present some concepts used in developing our proposed architecture and building the evaluation framework.

A. 5G Core & Service-Based Architecture

Starting with 3G, the mobile networks are separated in two main parts, namely: i) the Access Network (AN), and ii) the Core Network (CN). As the name suggests, the AN enables the User Equipment (UE) to gain access to the service provider's network. The AN can be described as a pool of distributed entities that provide coverage for the UEs, a task which in the case of Radio Access Networks (RAN) is achieved by the Base Stations (so-called gNBs in 5G).

For the UEs to be able to register with the operator and then communicate with other UEs or exchange data with services in the Internet, the AN needs to be connected to the CN. Starting with Release 14 [23], the CN implements the concept of Control and User Plane Separation (CUPS), similar to the one applied in SDN. In terms of control plane functionalities, the CN offers authentication, mobility and session management, policy control, network slicing, etc. Additionally, its user plane which consists of the User Plane Function (UPF) offers data tunneling, routing and forwarding, policy enforcing, etc.

5G Mobile Networks are envisioned to enable new use-cases in the areas of enhanced Mobile Broadband (eMBB), massive Machine-Type Communication (mMTC) and Ultra-Reliable and Low-Latency Communications (URLLC) [24]. Therefore, the CN should be designed while considering scalability, flexibility and automation. Being the main organization tasked to develop mobile communication specifications, 3GPP first introduced a standardized version of the 5G Mobile Core Architecture (5GC) in Release 15 [5], supporting several architectural principles [25], such as:

- *Modularized Network Functions* - The partial adoption of CUPS principles and deployment of 4G EPC entities as Virtual Network Functions (VNFs), meant that the future generations of CN could move away from the monolithic-based design of NFs. Therefore, the 5GC is based on highly modularized and distributed NFs that rely on inter-NF communication to serve the control plane procedures.
- *Adoption of cloud-native and Web-scale technologies* - At the core of this principle reside concepts such as: i) agile development, ii) microservice architecture, and iii) container orchestration. An agile development process brings a lot of flexibility and lowers the time-to-market for new 5GC NFs and features. Additionally, a microservice architecture and container orchestration technologies aid the agile development style and enable a more fine-grained orchestration and scaling of the workloads.
- *Distribution of processing power to the edge of the network* - By adopting a distributed architecture, the operators have now the flexibility to orchestrate their CN not only at the central office, but also at the Edge of the network. The processing capacity of the Edge can now be exploited, resulting in lower delays and less burden on the transport links connecting the Edge to the Core.

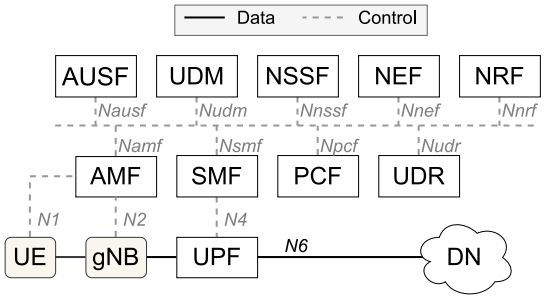


Fig. 1. 5G Core Service-Based Architecture as introduced by 3GPP.

These architectural principles led to the introduction of the 5G Service-Based Architecture (SBA), shown in Fig. 1. This architecture further decomposes the legacy EPC NFs such as MME with its functionalities being split between the Access and Mobility Function (AMF), Session Management Function (SMF) and Authentication Server Function (AUSF), as well as introduces new NFs such as Network Repository Function (NRF) or Network Slice Selection Function (NSSF). In EPC, the number of NFs was small and point-to-point interfaces were implemented for the communication between them (e.g., MME and Home Subscriber Server [HSS]). However, the situation changes greatly in 5G where the highly modularized and distributed SBA necessitates a simplification of inter-NF communication mechanisms. To address this, Service-Based Interfaces (SBIs) are introduced for the inter-NF communication. Similar to the microservice architecture, SBI is a concept borrowed from the IT applications domain. The main idea behind it is that each 5GC NFs exposes its services through REST interfaces that are defined in standardized OpenAPI [26] descriptions. During start-up the NFs register their services to the NRF, where other NFs can discover and consume them by sending HTTP/2 requests. Overall, the adoption of SBIs marks a significant shift in how the CN is designed and operated, distinguishing it from the previous generations.

B. State Management in 5GC

In Section III-A, we mentioned that the adoption of cloud-native principles and technologies is one of the main pillars of the 5GC design. As such, the deployment of NFs in cloud environments is performed using lightweight containers distributed among the available hardware resources. However, aiming to optimally schedule services in the infrastructure, cloud orchestration tools can decide at any time to terminate a container, move them into new machines or horizontally scale the deployed instances. Moreover, containers are not inherently designed to be highly reliable and may fail at any time.

A *stateful* application maintains locally the necessary context for its correct operation. In case the application crashes or is terminated, the information is lost and hence, affects directly the offered services. Therefore, the absence of guaranteed liveliness for a given instance makes the deployment of stateful applications unfeasible. In comparison, a *stateless* application is designed by separating the processing logic from the state database (DB). When the application needs context information, it queries the DB and upon finishing the processing, it updates the information in the DB. Stateless services can be scaled-out when the input load increases, as processing is not bound to any specific instance. In contrast, stateful services require traffic to be routed always to the instance which has the context information, hence requiring more careful lifecycle management than the stateless and ephemeral instances.

Aiming to support stateless deployments of 5GC, 3GPP introduced the Unstructured Data Storage Function (UDSF) to be used as a state DB. Statelessness in 5GC can be implemented at a *procedural* or *transactional* level [21]. A transaction can be defined as a single interaction between two NFs (e.g., request/response), and a procedure comprises a set of transactions. In the case of a procedural stateless NF, the information is exchanged only at the beginning and the end of the procedure. On the other hand, a transactional stateless NF communicates with UDSF for each individual transaction. While it is more fine-grained, transactional statelessness introduces a lot of overhead. Due to their design, some NFs cannot extract information regarding the ongoing procedure, making transactional statelessness the only implementable option. Nonetheless, statelessness is a feature and as such it is up to the developers and operators to decide *what* information to include as part of the state and *how* to store/retrieve it, as seen fit for their deployment. For more information on our implementation of statelessness in 5GC, refer to Section IV-A.

C. Free5GC

Free5GC [27] is one of the first open-source implementations of the 5GC SBA. In its early versions, it started as an extension of the NextEPC [28] towards 5GC by migrating MME, Serving Gateway (SGW), and Packet Data Network Gateway (PGW) to AMF, SMF and UPF. New 5GC NFs were added gradually, together with a full migration to SBIs according to the 3GPP and OpenAPI specifications. Starting from v3.0.0, Free5GC offers a fully operational 5GC SBA implementation compliant with Release 15 and a viable platform for 5GC evaluations and feature-testing [29].

Free5GC is developed following an agile process, where each of the NFs are developed separately, thus allowing for fast improvements and integration of new features. The Free5GC system offers: i) control plane NFs including AMF, SMF, AUSF, NRF, NSSF, Policy Control Function (PCF), Unified Data Management (UDM) and Unified Data Repository (UDR); ii) a softwarized User Plane Function (UPF). The control plane functions are implemented using Go [30] which allows building scalable applications with a very small memory footprint. Additionally, the NFs can be easily containerized because their code can be compiled into static binaries. The UPF, on the other hand, is implemented in C language and serves as a good prototype for testbed deployments of 5GC.

Alongside the NFs mentioned above, a MongoDB [31] instance is deployed. It serves as a backend database for NRF and UDR to store NF-related and UE-related information such as policy data, authentication keys, authentications status, etc. However, its purpose is not to enable any form of stateless

deployment for the control plane NFs. The UE context and the NFs' self-context is still maintained locally.

Other alternatives exist that implement the 5G SBA such as Open5GCore [32], OpenAirInterface 5GCN [33] and Open5GS [34]. However, Free5GC is one of the first, open-source and well-maintained projects and therefore it is considered for this work. We have also had the chance to contribute to its source-code with a feature that enables the NFs to advertise Kubernetes Service domain names when registering to NRF, thus making the deployment of Free5GC in a cloud environment easier [9].

D. Cloud-Native Orchestration

Public and private cloud environments conveniently offer flexible orchestration, high resiliency and high scalability. Thus, by leveraging cloud-native orchestration tools, telecom service providers benefit from: i) on-demand service provisioning and autoscaling, ii) cost-efficient operation and management, and iii) faster time-to-market for new services.

Until recently, the development process of new applications followed a monolithic approach, where despite the logical modularity, the application is packaged and deployed as a single artifact. However, the monolithic architecture poses some major drawbacks, such as:

- Applications may become too large and complex thus affecting the start-up time.
- Difficulties in scaling because different modules may have conflicting resource requirements.
- Difficulties in adopting new technologies as changes in languages or frameworks will affect the entire application.

This does not mean that monolithic applications cannot be deployed in cloud environment, though their architecture may become a barrier if the developers want to leverage the benefits of cloud environments. Therefore, at the core of cloud-native deployments, resides the concept of *microservices*. Contrary to monolithic applications, an application based on the microservices architecture is split into multiple small applications or services. Each microservice implements a specific set of functionalities and inter-communication is achieved by exposing and consuming APIs. A typical deployment of such architecture consists of the separation of the communication stack, processing logic and storage management.

Containerization is a lightweight virtualization technology that abstracts the underlying physical resources from the applications and provides process-based isolation such that microservices do not interfere with one another, while still sharing the same operating system kernel. However, microservice applications can quickly become too complex to orchestrate and manage as more instances are deployed to handle the incoming load or provide additional features. Therefore, containerized workloads and services are generally managed through container orchestration tools. Kubernetes (K8s) [22] is the most used framework among public and private cloud operators for managing their clusters. A K8s cluster consists of a set of nodes (bare-metal or VMs), which act either as *Master* or *Worker*. To create the cluster, the K8s control plane applications are initialized at the Master node and then Worker nodes are added as required by the operator's needs. During registration, the Workers expose their resource information to the Master which has global knowledge over the entire cluster and keeps track of the Workers' states. When new workload requests come from the operator, the Master handles their scheduling among the available Workers. In a cloud environment, containers are considered to be ephemeral, meaning that they can fail or be destroyed at any time to match the desired state of the cluster. In such cases, the advantages of tools like K8s become apparent as it can restart crashed applications, reschedule workloads when node failures occur, perform rolling updates, etc.

Some K8s concepts used in this work are explained below [35]:

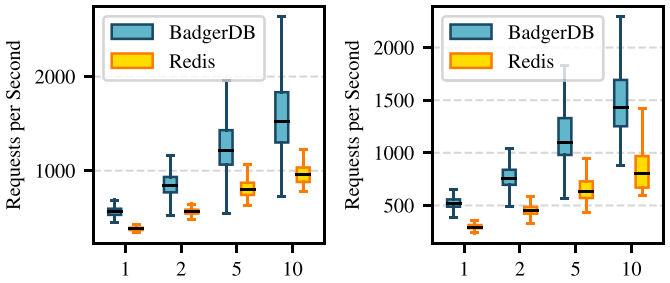
- *Pod* - Is a group of one or more containers and it is the smallest unit that can be created, deployed and managed with K8s. Within a Pod, the storage and network resources are shared.
- *DaemonSet* - Is used to ensure that a copy of a Pod runs in all (or some) of the nodes. Use cases include deployment of monitoring and log-collection daemons.
- *Service* - Communication between applications in a K8s cluster cannot always rely on the Pods' IP addresses because they may not be known at initialization time or they may change during runtime. To overcome this, *Services* are introduced as an abstraction mechanism to expose the application running on a set of Pods. Services have their own IP address and DNS name and can load-balance the incoming traffic among the set of Pods.

IV. System Design and Implementation

In this section, details about the implementation of the Per-Procedure Stateless 5G (PP5GS) system are given. First, the implementation of UDSF for our stateless 5GC is presented. Next, the architecture of Per-Procedure Network Functions (PPNFs) is introduced and the implementation steps are summarized. During the development phase, we follow Continuous Integration and Continuous Development (CI/CD) practices. We have set up a CI/CD pipeline into our `git` repository that compiles each NF after its implementation changes. Docker images are then built from the new binaries and pushed to our private container registry. The stored images are easily accessible during deployment and the configuration files ensure that always the latest versions are chosen.

A. Stateless Free5GC

As part of the contributions of this work, we have implemented a stateless system by separating the context from the processing logic. 3GPP Release 15 includes a specification for the Unstructured Data Storage Function (UDSF) which exposes a data repository service that accepts arbitrary formed binary payloads, the so-called *blocks*. Clients (other NFs) can then store application state at the UDSF in any serialization format. One or more blocks belong to a record, which can store additional metadata such as Time-To-Live (TTL) duration. Additionally, a schema description can be attached to



(a) GET record operations. (b) PUT record operations.

Fig. 2. Comparison of throughput achieved by BadgerDB and Redis for a) *read* and b) *write* operations.

every record to achieve self-describing data records. UDSF exposes two hierarchical levels that allow for namespacing and sharding of records. For ease of implementation, we choose to serialize the application state in the JSON text format. The block payloads are handled transparently and the received bytes are returned unmodified. Being unstructured, UDSF's implementation is backed by a key-value database. The key is constructed from the hierarchical and unique record identifiers. Every record is stored as a single value to ensure safe TTL expiration and atomic updates. One choice for the database backend is the embedded BadgerDB database [36]. It operates only in-process and is not distributed between instances of UDSF. Another choice is the key-value store Redis [37]. Here the UDSF connects to a remote database server and acts as an SBA proxy for the 5GC. Fig. 2 shows a benchmark where we compared the two implementations over a wide range of payloads sampled from the different client NFs. While we observe acceptable performance for both options even under increasing concurrency, the embedded BadgerDB can perform considerably more *read* and *write* operations. Therefore, in order to ensure consistency we deploy a single UDSF instance with BadgerDB as backend database.

In addition, some of the 5GC NFs need to be modified to enable statelessness. NRF and UDR reuse the already existing MongoDB connection to store subscriptions and do not require modifications. Moreover, NSSF does not store any UE-specific application state. For the AMF, many small NGAP messages are exchanged in quick succession per procedure, and therefore, procedural statelessness is a reasonable choice. It loads the context to handle a UE at the beginning of a procedure and stores it back on completion. To help with development and testing, AMF will always perform an initial Registration procedure and not query UDSF for any existing UE context during this procedure. Following the same logic, the UE context is not deleted from UDSF after performing a UE-originating Deregistration procedure. Technically, UDSF allows a get-and-forget operation, where the record is returned and deleted in the same request.

For the remaining NFs, namely AUSF, PCF, SMF, and UDM, we implement transactional statelessness from the perspective of the HTTP API. On request, the NF will try to load the UE context from UDSF and, before returning a response, it will store it back. Using this naive implementation, we ensure the context is always up to date and consistent. This, however, introduces overhead, as, e.g., UDM is frequently used by the other NFs and causes multiple exchanges with UDSF. While UDSF supports the HTTP entity tag header that allows the requesting NF to indicate the last known state and avoid unnecessary communication, this mechanism is not yet implemented by the client NFs. In SMF, the transactional statelessness also covers any communication with the UPF as part of the request.

TABLE I
INVOLVED NFS IN THE EXECUTION OF CONTROL PLANE PROCEDURES

Procedure \ NF	AMF	SMF	AUSF	PCF	NRF	NSSF	UDM	UDR
Registration	✓		✓	✓	✓		✓	✓
PDU Sx. Est.	✓	✓		✓	✓	✓	✓	✓
PDU Sx. Rel.	✓	✓						
Deregistration	✓			✓			✓	✓

B. Per-Procedure Network Functions

With the high degree of functional decomposition introduced by SBA, the execution of control plane procedures spans multiple NFs, each performing a specific task. In Table I, the involved NFs in each of the considered control plane procedures are shown. Between these NFs, multiple messages are exchanged using a request/response or subscribe/notify mechanism. While there are other control plane procedures as well, the four selected procedures are representative given that they have different communication profiles. The Registration and PDU Session Establishment procedures trigger complex Service Function Chains (SFCs) where 6 and 7 NFs are involved, respectively. On the other hand, the PDU Session Release and Deregistration procedures rely less on inter-NF communication and the logic is concentrated in fewer NFs. UPF is also involved in some of the procedures, however, our focus is only on the control plane NFs.

The main idea behind the proposed PP5GS system is to break the tight inter-NF dependencies through the introduction of Per-Procedure NFs (PPNFs). These new NFs are self-contained, meaning that all the necessary logic for serving a specific control plane procedure is integrated in the source-code of the NF. We develop PP5GS using the source-code of stateless Free5GC v3.0.6 (see Section IV-A) as the basis for the new NFs. First, all the involved NFs are identified and the observations are validated with the relevant 3GPP specifications. Due to their functions as service-discovery mechanisms or database abstraction layers, we do not integrate NRF, UDR and UDSF in the new PPNFs. Their implementation is dependent on the backend database used by the operator and therefore integrating them into multiple PPNFs is not feasible from an implementation perspective. For example, the source-code of the NF serving Registration procedure (*RegNF*) only contains logic that in an SBA system is distributed among AMF, AUSF, PCF and UDM. Next, each processing block that is executed during the given procedure needs to be identified. To achieve this, we track the requests that are initiated inside 5GC and leverage the SBIs that the involved NFs expose through their HTTP/REST server. By inspecting the inter-NF

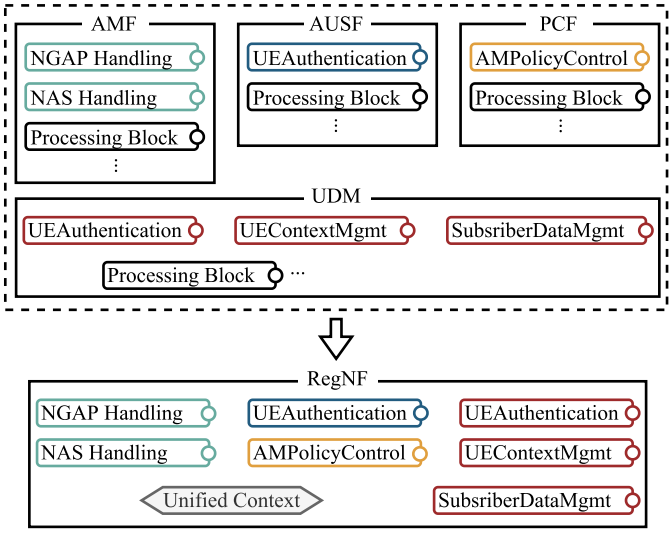


Fig. 3. Integration of processing blocks from different NFs in a single Registration PPNF. The blocks highlighted with colors are the ones executed during the Registration procedure.

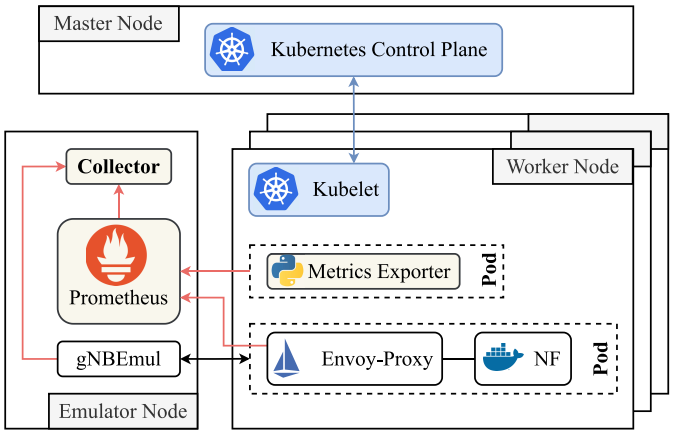


Fig. 4. Overview of the framework setup for the evaluation of the different 5GC systems. The cluster is orchestrated using Kubernetes.

traffic that is generated in 5GC, it is possible to extract the URI information of the destination endpoints for each request, leading us to their *callback* functions.

In the integration phase, AMF is considered as the base NF and its functionalities are extended with the other processing blocks. To better understand the architecture of PPNFs, we refer the reader to Fig. 3 where RegNF is shown. Each of the highlighted processing blocks do not represent a single callback function, but rather a logical group of functions. RegNF retains the AMF's NG Application Protocol (NGAP) and Non-Access Stratum (NAS) communication handling logic necessary to process the packets coming from UE and gNB, and in addition the following endpoints are integrated: i) `/nausf-auth` is migrated from AUSF, ii) `/npcf-am-policy-control` is migrated from PCF, and iii) some functions from the `/nudm-ueau`, `/nudm-uecm` and `/nudm-sdm` are migrated from UDM.

Moreover, a unified UE context is built such that it includes all the information previously split among the different NFs. Particular attention is given to any duplicated information in order to avoid redundancy and unnecessary overhead. Since in PP5GS there will be different NFs handling different procedures, it is necessary that at least procedural statelessness is implemented. Therefore, once a PPNF finishes serving a procedure, it serializes the UE context and state and sends it to UDSF which maintains the latest versions. The local context is deleted to avoid wasting memory resources unnecessarily. When a following procedure is initiated, the responsible PPNF queries the information from UDSF before proceeding to serve the request.

Using the methodology described above, in this work we have successfully developed *RegNF*, *PDUEstNF*, *PDURelNF* and *DeregNF* to execute Registration, PDU Session Establishment, PDU Session Release and Deregistration procedures, respectively, as part of our proposed PP5GS solution.

V. Performance Evaluation

In this work, we mainly focus on the performance assessment of our proposed architecture in a small scale private cloud-native environment. As shown in Fig. 4, we set up a testbed composed of 11 bare-metal nodes (machines) and use Kubernetes as the orchestration tool. The machines are interconnected using a 1Gb switch, thus creating an isolated environment and avoiding network interference.

In this testbed we distinguish between three types of nodes:

- *Master Node* - It is responsible only for running the K8s control plane and orchestrating the deployment of 5GC systems. No additional workload is scheduled on this node.
- *5GC Worker Node* - The baseline deployment mode that we consider is a fully distributed one where each NF is scheduled on a separate node. This mode is selected for two main reasons: i) to avoid multiple NFs competing for the same physical resources, and ii) ensure comparable communication conditions as inter-NF traffic will traverse the same infrastructure in all scenarios. In addition to the 5GC NFs, an Envoy-Proxy [38] instance is deployed as a sidecar container as part of Istio [39] service-mesh. It facilitates traffic routing and management and allows for some metrics collection as well. The considered deployments consist of at most 9 control plane NFs, thus we deploy 9 worker nodes in the cluster. The cluster of workers is homogeneous and made of DELL OptiPlex 9020 workstations equipped with an octa-core Intel i7-4770 CPU running at 3.40 GHz and 16 GB of RAM.
- *Emulator Node* - This node is used to host the gNB & UE Emulator (gNBEmu) application. We develop this application for the purpose of generating scalable control plane traffic, and evaluate the performance of the 5G systems. For more information on the implementation details and capabilities of gNBEmu, please refer to Appendix A. Additionally, it hosts *Prometheus* and *Collector* instances which are necessary for the measurements setup (see Section V-A).

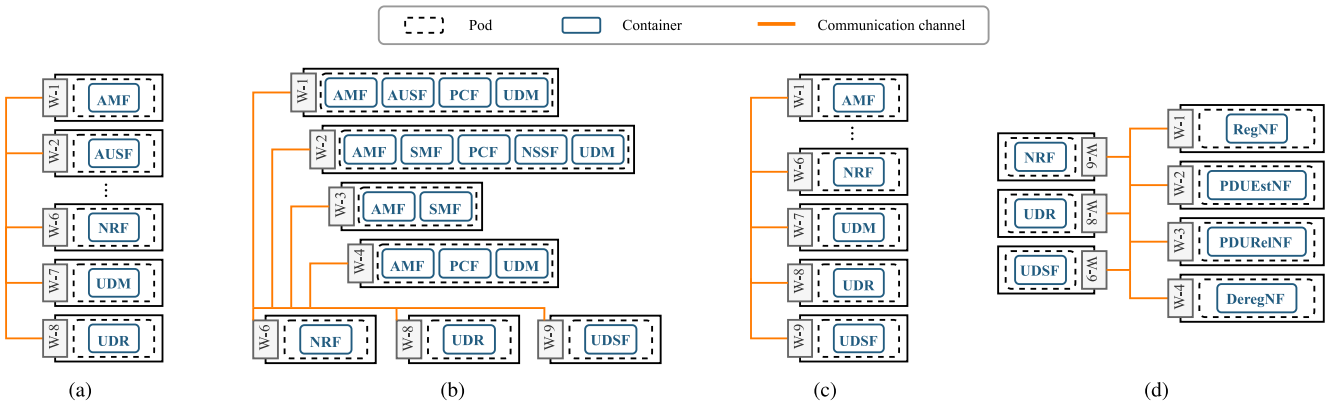


Fig. 5. Overview of the evaluated systems: *a)* Stateful Free5GC, *b)* Stateless Free5GC, *c)* Procedure-Pods, and *d)* PP5GS. The hardware infrastructure is the same for all the systems.

All the nodes in the testbed run Ubuntu 18.04.4 LTS with kernel version 5.0.0-23-generic. The installed container-runtime is Docker v20.10.5. Kubernetes v1.22.1 is deployed with Calico [40] being used as the Container Network Interface (CNI) plugin.

Lastly, some of the control plane procedures that we evaluate necessitate the deployment of UPF. Therefore, we have developed a lightweight UPF emulator (UPFe) that can handle a high rate of control plane traffic. In Appendix B, you can find a detailed description on the implementation of UPFe.

TABLE II
SUMMARY OF INPUT PARAMETERS

Parameter	Value
Control plane procedure	• Registration • PDU Session Establishment • PDU Session Release • Deregistration
NF deployment	1 Pod/Node
Number of new UEs/sec	25, 50, 100, 200, 300
IAT for new UEs	3 ms
Measurement duration	45 s
Campaigns per scenario	8

A. Measurement Setup

For the purpose of evaluating the improvements that PP5GS brings, we consider the following Systems Under Test (SUTs):

- *Stateful Free5GC* - This is the default implementation where we only optimize SCTP-related parameters (see Appendix A). Each of the NFs is deployed in a separate Node, as illustrated in Fig. 5a. In this work, this deployment is considered as the baseline system.
- *Stateless Free5GC* - It is built on top of Stateful Free5GC, by implementing statelessness as described in Section III-B. As shown in Fig. 5c, in this system we additionally deploy the UDSF instance which is shared by the NFs to store their application state/context.
- *Procedure-Pods* - In this system, we consider *slicing* 5GC based on the control plane procedures, as illustrated in Fig. 5b. To achieve this, we initialize multi-container Pods where all the involved NFs (see Table I) are deployed in a single Pod. This deployment requires the use of stateless NFs since more than one instance/NF are deployed, e.g., one AMF instance per *slice*. Each of the Procedure-Pods is deployed in a separate Node. The inter-NF communication does not leave the Pod, with the exception of traffic destined to NRF, UDR and UDSF which are shared between all NFs and deployed once for the entire cluster.
- *PP5GS* - Instead of the standardized 5GC NFs, in this system we deploy the PPNFs developed with the aim of breaking the inter-NF dependencies during the execution of control plane procedures. Each of the PPNFs is deployed in a separate node, while sharing the functionalities of NRF, UDR and UDSF in the same manner as Procedure-Pods. This deployment is illustrated in Fig. 5d.

In addition to the above-mentioned systems, we deploy a metrics collection framework. First, we have written a Python script which finds the NFs that are running in a Node and then collects resource utilization data about them. These data points are then wrapped as a gauge object and exposed through the Prometheus API [41]. The *Metrics Exporter* script is packaged into a Docker image and deployed as a DaemonSet in each Node of the cluster. The Prometheus instance that runs in the Emulator Node (see Fig. 4) is configured to periodically scrape the endpoints every 1 s. In addition, we deploy a *Collector* application. The Collector is a data sink and it implements a HTTP server to facilitate the metrics reporting process. Furthermore, the Collector queries the NFs' logs and K8s cluster state to facilitate debugging.

In Table II, we summarize the input parameters used during the evaluation. We asses the performance of the proposed PP5GS for four different control plane procedures: i) Registration, ii) PDU Session Establishment, iii) PDU Session Release, and iv) Deregistration. Furthermore, we consider different rates of input traffic with the number of new UEs/s taking the following values [25, 50, 100, 200, 300], and each scenario running for 45 s. We define an interarrival time (IAT) of 3 ms for new procedures, following a uniform distribution. This value is used only to spread the initialization of new UEs over a 1 s interval, and it does not affect the UEs/s values defined above. Overall, 8 campaigns for each scenario

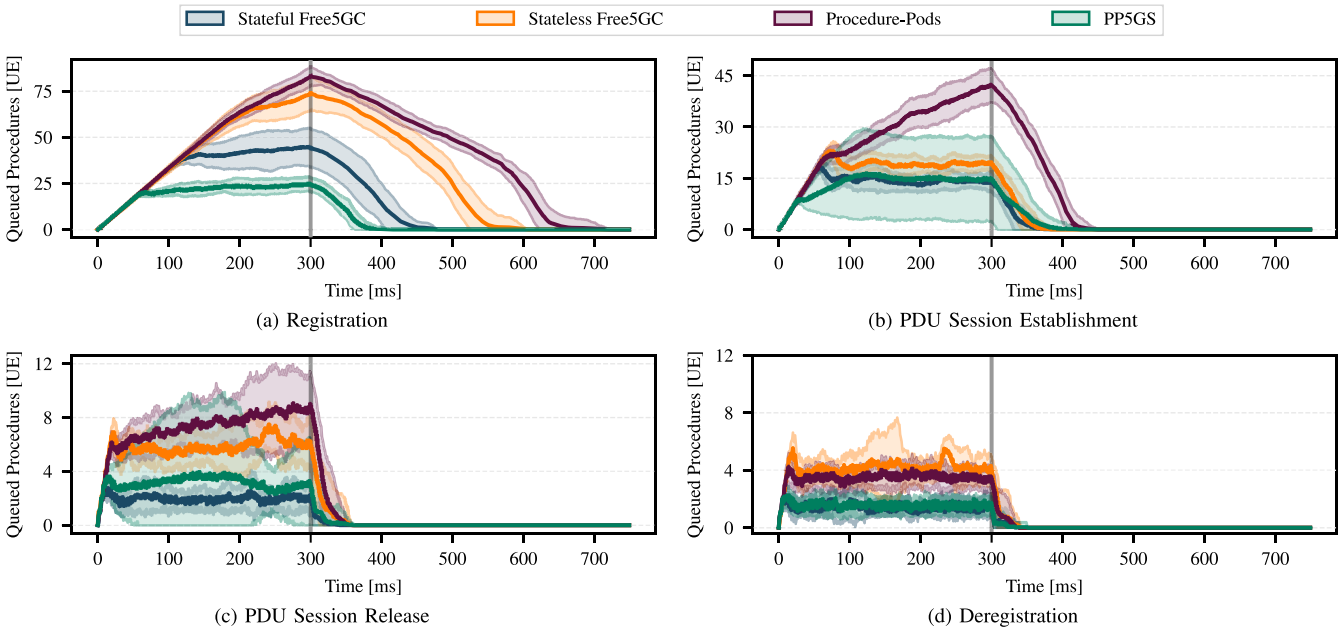


Fig. 6. Number of UEs that have started a control plane procedure within 1 ms for a scenario with 100 initiated UEs/s. With a uniformly distributed IAT of 3 , increases in the queue are expected until $t_1 = 300$ ms. The darker lines show the average values while the lighter areas denote variance based on the standard deviation. The results confirm that gNBEmu is able to handle large volume of traffic generation, and they reveal a potential bottleneck in the Stateless Free5GC and Procedure-Pods systems when input rates are higher than 100 UEs/s.

are run in a fully automated manner and the measurements are then aggregated.

In the following subsections, we will introduce the Key Performance Indicators (KPIs) and present the obtained results from our evaluation.

B. Emulator Performance

We emphasize that the control plane traffic generation differs from that on the data plane since a control plane procedure requires exchanging multiple messages between RAN and 5GC (for more information refer to Appendix A). Therefore, we first assess the performance of gNBEmu to confirm that it can indeed generate traffic according to its configuration file description. The evaluation is performed for all the SuTs and four independent measurements that run for 45 s are executed for each control plane procedure. The input traffic consists of 100 UEs/s with an IAT of 3 ms. Therefore, within 1 s it is expected that no new procedures are initialized after $t_1 = 300$ ms.

First, we collect the procedures' start and end timestamps from the gNBEmu. In post processing, we then calculate the number of UEs that are concurrently being served by the system. In Fig. 6 we show the average number of queued procedures (UEs) in the system within the 1 s interval for all the considered control plane procedures. When comparing the results of different procedures, we observe that the Registration procedure (shown in Fig. 6a) exhibits the highest number of queued UEs for all the systems. This is expected as it is the most complex procedure in terms of inter-NF communication, hence taking more time to finish each execution. Nonetheless, the gNBEmu correctly initiates new procedures in accordance with the traffic model and starting from $t_1 = 300$ ms the number of queued UEs drops (i.e., there are no new procedures initialized after this point).

Overall, PP5GS exhibits the lowest number of queued UEs in the system for three out of four procedures. Only in the PDU Session Release shown in Fig. 6c, we observe that Stateful Free5GC outperforms PP5GS. On the other hand, Stateless Free5GC and Procedure-Pods exhibit poorer performance, especially in the case of the Registration procedure. Out of the total 100 triggered UEs in 1 s, at $t_1 = 300$ ms there are 73 and 82 queued UEs for the Stateless Free5GC and Procedure-Pods, respectively. Consequently, the execution of all queued Registration procedures finishes at $\sim$ 600 ms for Stateless Free5GC and $\sim$ 700 ms for Procedure-Pods. Therefore, we expect that for scenarios with 200 and 300 UEs/s the execution will not be completed within the 1 s and some of the procedures will be carried to the next timeslot, leading to an overload in the 5GC.

C. CPU Utilization

In this work, we consider CPU utilization to be one of the main KPIs for comparing our proposed architecture with the other SBA systems. The main reason behind this is the importance of efficient resource utilization in cloud-native 5GC, as this inherently leads to better scalability and facilitates edge offloading of 5GC NFs.

In our measurement setup, the Metrics Exporter script collects CPU Utilization data every 1 s for all the deployed NFs. The average CPU utilization values with respect to the number of new procedures per second are shown in Fig. 7. Stacked bar plots are chosen to better illustrate the cumulative system CPU utilization. In Section IV-B, we explained that the functionalities of NRF, UDR and UDSF are not integrated in the PPNFs.

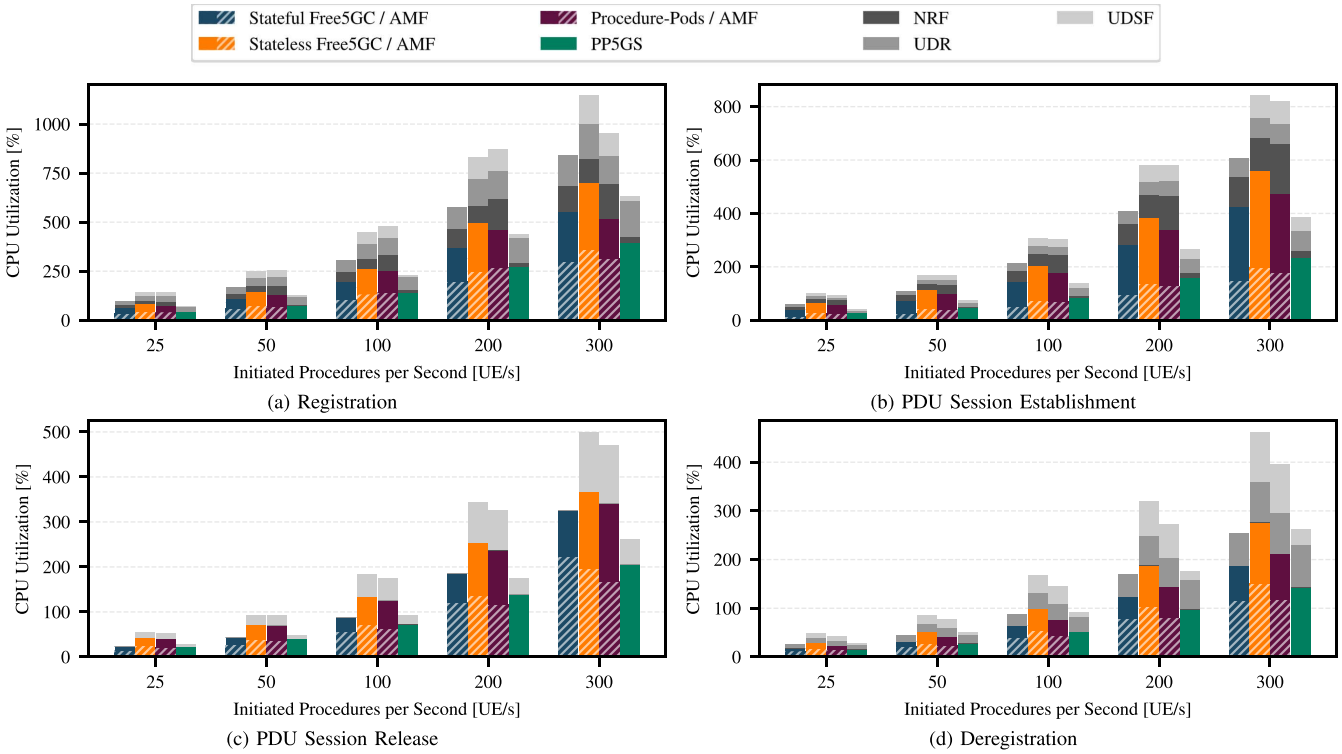


Fig. 7. Average CPU utilization of 5GC NFs w.r.t. the number of initiated procedures per second, for all the considered control plane procedures. The hatched areas represent AMF's utilization in each of the systems where it stands as a separate NF. The solid colored areas represent the utilization of the rest of NFs in the chain, which in the case of PP5GS are integrated together. Utilization of NRF, UDR and UDSF is shown separately since they are deployed as separate Pods in all systems.

Therefore, we illustrate their CPU utilization in grayscale, while the rest of the involved NFs are shown in colors. For the Stateful Free5GC, Stateless Free5GC and Procedure-Pods, the utilization of AMF is highlighted and represented in dashed lines.

For all the considered procedures and input traffic configurations, the Stateless Free5GC system exhibits higher CPU utilization compared to the Stateful Free5GC. During the execution of more complex procedures, Stateless Free5GC consumes on average 43% and 49% more resources for the Registration and PDU Session Establishment procedures, respectively. The difference becomes bigger for the PDU Session Release and Deregistration procedures where, respectively, 98% and 89% more resources are consumed. A major contributor to this increase is the UDSF, and as can be seen in Fig. 7c and Fig. 7d, it comprises a big chunk of the overall CPU utilization. Nonetheless, an increase of the consumed resources can be observed also by comparing the colored bars. This is a result of the fact that in the stateless system the NFs need to perform additional processing such as serialization/deserialization of their context to communicate with UDSF.

Similar results are observed also for the Procedure-Pods system due to the fact that the same implementation of NFs is used as in the Stateless Free5GC system. However, compared to Stateless Free5GC, we observe an increase of the NRF's utilization by ∼55% during Registration and ∼46% during PDU Session Establishment. The reason for this is the higher number of deployed NFs while there is only one NRF instance in the Procedure-Pods scenario. Moreover, an interesting behavior occurs during Registration for the scenario with 300 UEs/s. Unlike the previous values, Procedure-Pods setup exhibits lower utilization compared to Stateless Free5GC. This behavior can be explained using the observations derived from Fig. 6a, where we concluded that for high input rates, the execution may not always be completed within the expected 1 s time frame, leading to a bottleneck in 5GC. Therefore, in this scenario the high number of unfinished procedures has caused AMF to bottleneck and consequently produce less traffic towards the other NFs in the chain.

Our proposed PP5GS system exhibits the best performance in terms of efficient resource utilization. For the Registration procedure, on average it requires 26% and 42% less resources compared to Stateful Free5GC and Stateless Free5GC respectively. The efficiency increases further during PDU Session Establishment Procedure, with an improvement of 34% compared to Stateful Free5GC and 55% compared to Stateless Free5GC. For the less complex procedures, PP5GS' performance seems to be on-par with Stateful Free5GC which is again a very good performance considering the added feature of statelessness. Compared to Stateless Free5GC, ∼45% less resources are required for both procedures.

D. Procedure Completion Times

One of the goals we aim to achieve with our proposed PP5GS is the reduction of latency or Procedure Completion Time (PCT). While generally the goal in 5G is to reduce data plane latency, the fast completion of control plane procedures has a direct impact on the data plane. For instance,

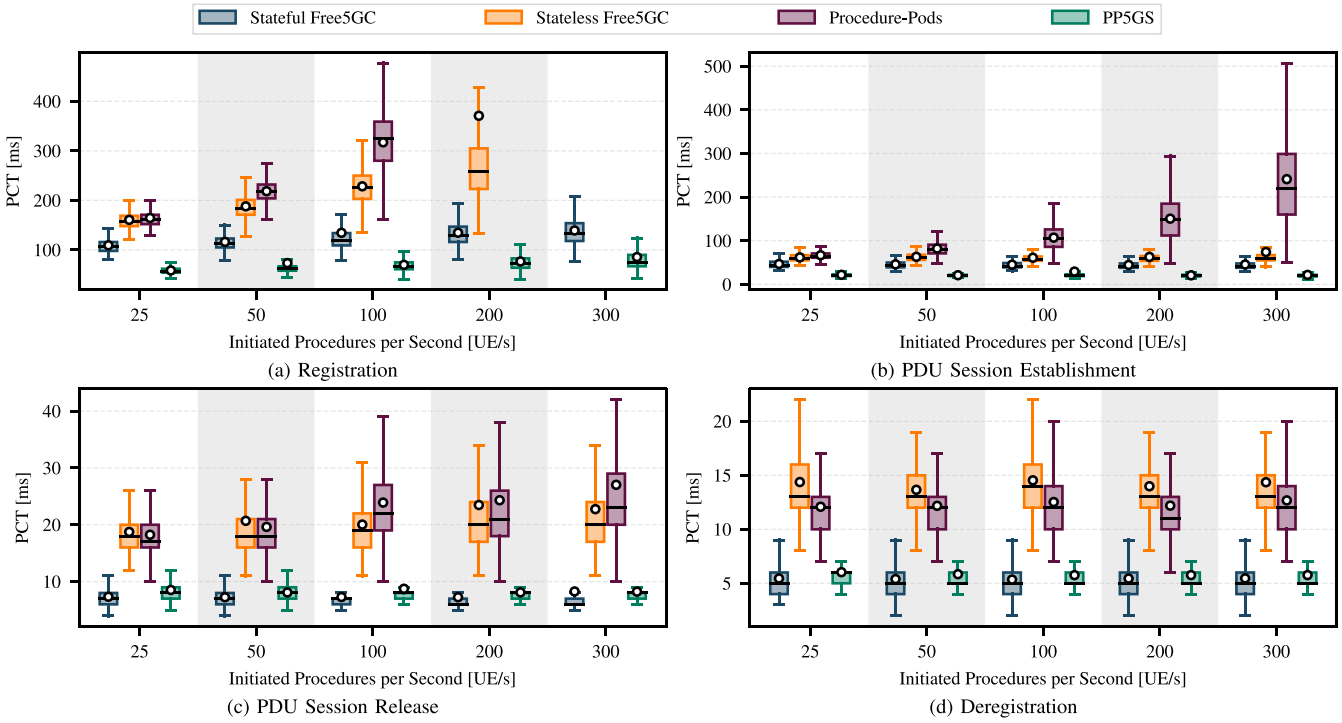


Fig. 8. Procedure Completion Times w.r.t. the number of new initiated procedures per second for all the considered control plane procedures. Each box represent data collected over 8 campaigns, each executing for 45 s. For better visibility, we have omitted the outliers from the plots while still considering them when calculating the mean values.

the faster 5GC manages to complete Registration and PDU Session Establishment, the sooner can UEs start transmitting in the data plane. Therefore, we consider PCT to be the main KPI denoting the performance of 5GC.

As denoted by its name, PCT is defined as the time needed to complete the execution of a control plane procedure. Having full control over gNBEmu allows us to collect timestamps and directly calculate this metric. The four procedures considered in this work are all UE-initiated and the first timestamp is taken right before gNBEmu forwards the first message to 5GC. Below we give some more details regarding the messages that trigger the timestamping operation for each procedure:

- *Registration* - We consider the time between `Registration Request` and `Registration Complete` NAS messages.
- *PDU Session Establishment* - To mark the start of the procedure we use `PDU Session Establishment Request` and the timer is stopped after receiving the `PDU Session Establishment Accept` NAS message.
- *PDU Session Release* - `PDU Session Release Request` and `PDU Session Release Complete` are respectively the first and last SM NAS messages that trigger timestamping.
- *Deregistration* - The UE-initiated Deregistration procedure starts with a `Deregistration Request` NAS message and ends with a `UE Context Release Complete` NGAP message.

In all the procedures, except for the Registration Procedure, there are still some exchanged messages between 5GC NFs taking place even after we timestamp the end of the procedure. However, in this work PCT reflects the experienced processing time from the UE perspective and we are therefore not interested in including the execution time of the remaining processes.

The PCT comparisons for the four systems with respect to the number of new initiated procedures per second are shown in Fig. 8. The results collected for the Registration and PDU Session Establishment procedures confirm the enhanced performance of PP5GS compared to the baseline Stateful Free5GC. On average, PP5GS completes Registration 42% faster and PDU Session Establishment 50% faster than the baseline. Similar to what we observed in CPU utilization, statelessness of PP5GS has a higher cost for less complex procedures. Therefore, compared to the baseline, PDU Session Release (see Fig. 8c) and Deregistration (see Fig. 8d) take on average ∼12% and ∼8% longer. However, in terms of absolute values the difference is negligible being ∼1 ms.

On the other hand, Stateless Free5GC and Procedure-Pods exhibit very low performance compared to the other systems. Especially for the complex procedures, these systems scale very poorly with the increasing input rate. In the scenario with 300 UEs/s performing the Registration procedure, average PCTs reach 521 ms and 11.48 s for the Stateless Free5GC and Procedure-Pods, respectively. Therefore, we have excluded them from the plots as their performance is not comparable to the other high-performing systems.

PCT for other traffic patterns: In the evaluations presented above, we have defined a deterministic IAT of 3 ms for all the considered number of newly initiated procedures. To provide a better view on PP5GS's performance and how it compares

TABLE III
CONFIGURATION OF IAT DISTRIBUTION FOR DIFFERENT TRAFFIC PATTERNS

Configuration	IAT Distribution
A	Deterministic: 2.5 ms
B	Bimodal: 2 ms or 3 ms with probability 0.5 each
C	Multimodal: 1 ms, 2 ms, 3 ms or 4 ms with equal prob.

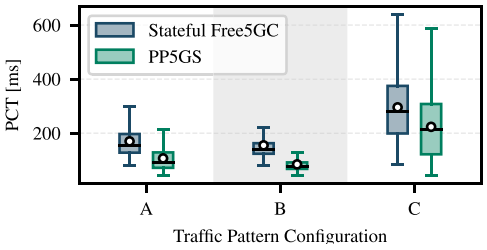


Fig. 9. Procedure Completion Times for the *Registration* procedure for 400 UEs/s with different traffic pattern configurations.

with the baseline, we define three different configurations of IAT distribution resulting in different control plane traffic patterns. These configurations are presented in Table III, and for each configuration we consider a scale of 400 UEs/s. In configuration *A*, we consider a deterministic IAT of 2.5 ms to accommodate the initialization of 400 UEs/s. In configuration *B*, we consider a bimodal distribution where the IAT between new procedures has a value of 2 ms or 3 ms where each option has a probability of 0.5. In configuration *C*, a multimodal distribution is considered where the IAT has a value of [1 ms, 2 ms, 3 ms, 4 ms], each with an equal probability of 0.25.

In Fig. 9, we show the PCT values for the Registration procedure for the proposed PP5GS system and the baseline Stateful Free5GC. The obtained results show that for the same number of UEs/s, the traffic patterns yield different results w.r.t. the PCTs. Nonetheless, PP5GS still outperforms the baseline for all the considered configurations.

E. Control Plane Communication Overhead

With the proposed PP5GS, we show the reduction of control plane communication overhead stemming from the adoption of SBA and the high functional decomposition. In PP5GS, the deployment of the self-contained PPNFs inherently reduces the exchanged traffic because: i) there is no inter-NF communication between the integrated NFs, ii) less requests need to be send to NRF for discovering the other NFs in the chain, and iii) less communication with UDSF since the entire system is procedural-stateless.

In order to evaluate this KPI, data regarding number of HTTP requests per second are collected by leveraging the Envoy-Proxy container (see Section V-A). The number of requests per second with respect to the destination NF for each of the control plane procedures are shown in Fig. 10. As for destination NFs we consider NRF, UDR and UDSF since they are deployed separately, and *"Other"* which refers to SBA NFs that are integrated into PPNFs. Measurements are collected for a scenario with 100 new procedures initiated every 1 s.

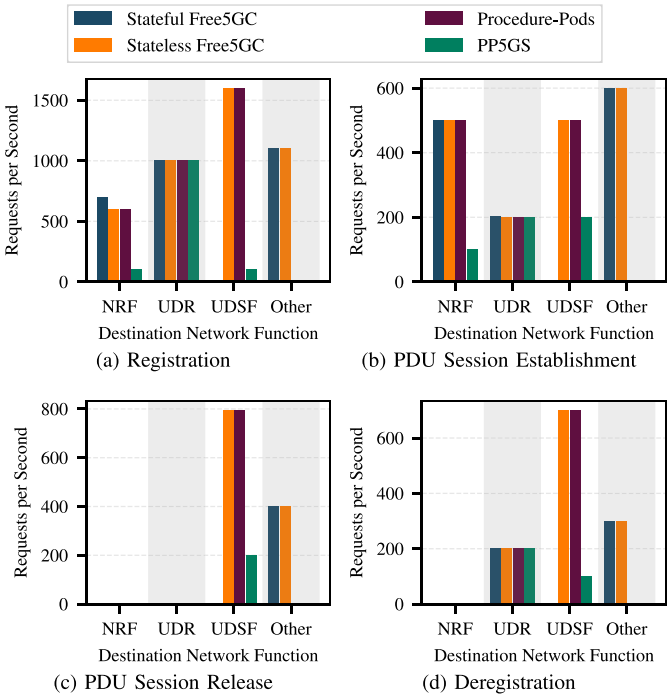


Fig. 10. Number of HTTP requests per second triggered during the execution of different control plane procedures w.r.t. the destination NF. The values are proportional to the number of UEs/s executing their procedures. Measurements are collected for the scenario with 100 new UEs/s.

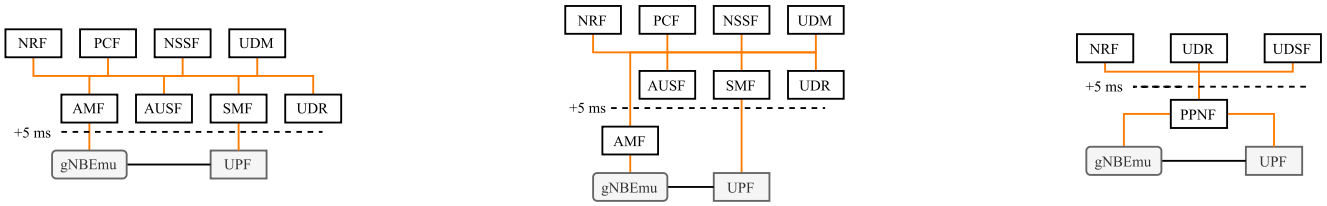
When comparing Stateless Free5GC and Procedure-Pods, the only difference is regarding traffic destined to *Other* NFs. Having a multi-container deployment in the case of Procedure-Pods means that the inter-NF traffic does not leave the Pod namespace, hence no traffic destined for *Other*. Nonetheless, in a more optimal deployment it would make sense for the Procedure-Pods system to avoid sending NF-discovery requests and instead use a static addressing scheme.

To compare the total number of requests generated in the systems, we sum the number of requests destined for each of the involved NFs. For the Registration procedure, PP5GS achieves a 57% reduction compared to Stateful Free5GC and 72% compared to Stateless Free5GC. During PDU Session Establishment, PP5GS generates 61% and 72% less requests compared to Stateful Free5GC and Stateless Free5GC, respectively. During the PDU Session Release and Deregistration procedures, PP5GS achieves a reduction of 83% and 75%, respectively, compared to Stateless Free5GC, while in comparison to Stateful Free5GC the reduction is 50% and 40%.

The measured average number of requests per second confirms the expected improvements as calculated by observing the inter-NF communication patterns. For all the procedures, PP5GS proves to be considerably more efficient in comparison to the other systems, therefore being a very good candidate architecture in reducing the control plane signaling overhead in 5GC.

F. Performance Evaluation for Edge Offloading Scenarios

Lastly, we evaluate the performance of PP5GS in scenarios where edge offloading of 5GC NFs is performed. While 5GC SBA offers great flexibility in orchestration and deployment,



(a) Centralized 5GCN. (b) Edge offloading of AMF. (c) Edge offloading of PPNF.

Fig. 11. Overview of the deployments considered for edge-offloading evaluations.

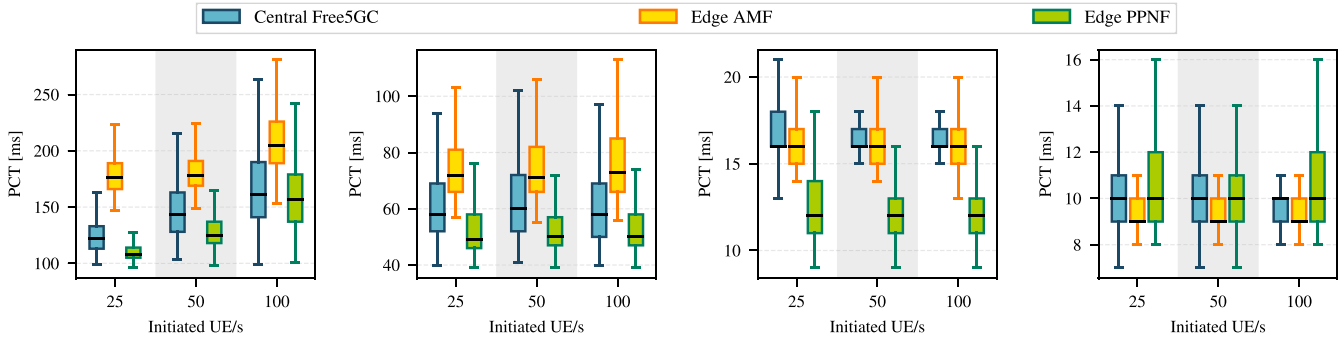


Fig. 12. Procedure Completion Times w.r.t. the number of new initiated procedures per second for all the considered control plane procedures. For the *CentralizedFree5GC* and *EdgeAMF* scenarios, the stateful implementation of Free5GC is used. Edge PPNF on the other hand uses the proposed PP5GS implementation. For better visibility, outliers are omitted from the plots.

the inter-NF dependencies during procedure execution may in fact be counter-productive. To confirm this behavior, we set up a testbed where a logical separation between the *Edge* and *Core* parts of the network is created. Using a tool such as Linux Traffic Control (`tc`), a 5 ms round-trip delay is injected to the interface connecting the edge node to the rest of 5GC. Then, we consider the following deployments for our evaluation:

- *Centralized 5GCN* where only gNBEmu and UPF run at the edge, while all control plane NFs are deployed in the *central* cloud as shown in Fig. 11a.
- *Edge AMF* where only AMF is offloaded to the edge in an attempt to increase the performance as shown in Fig. 11b.
- *Edge PPNF* where depending on the procedure the relevant PPNF is offloaded while NRF, UDR and UDSF reside in the Core as shown in Fig. 11c.

The boxplots of the PCT with respect to the number of new initiated procedures are shown in Fig. 12. It can be seen that the performance of 5GC during the execution of complex procedures drops when we offload AMF to the edge. The reason is that the number of messages exchanged between the AMF and the other NFs is higher than the amount of traffic exchanged between gNBEmu or UPF and the control plane NFs. Since every pair of messages exchanged encounters at least 5 ms of transmission delay, the PCTs increase considerably. This is however not the case for less-complex procedures where the processing logic is anyway mostly contained in the AMF and the difference between the two deployments is negligible. On the other hand, offloading a PPNF indeed mitigates the issue seen with offloading 5G SBA NFs. The median values of the obtained measurements for the Registration, PDU Session Establishment and PDU Session Release procedures are always lower than Edge-AMF. The only exception is observed during Deregistration where *Edge PPNF* exhibits similar performance to the other deployments. The reason is due to the number of messages exchanged between NFs deployed in the *Edge* and *Core* parts of the network being equal in all the considered deployments. Lastly, due to its architecture where processing logic is contained in a single PPNF, PP5GS is able to complete most of the procedures faster than the Centralized deployment as well.

VI. Discussion and Conclusion

In the evaluations we show that the benefits of a procedure-based architecture are manifold, especially for control plane procedures that require the involvement of many NFs during their execution. First, the CPU utilization evaluation shows that the integration of processing logic in a single NF reduces the overall needed resources, hence increasing the energy efficiency. Moreover, it enables deploying the 5GC NFs in the edge, which has limited resources compared to the central cloud. Compared to stateless deployments, PP5GS requires $\sim$42%$-$55% less CPU resources. Furthermore, we demonstrate that offloading single NFs to the edge in the case of an SBA deployment is in fact counter-productive for some control plane procedures. The high number of interactions with the NFs residing in the central cloud increases the completion time, a problem which PP5GS by design mitigates with the processing logic integrated in a single NF. Adoption of PP5GS allows for a reduction of at least 40% in control plane traffic.

Our evaluation shows that PP5GS is indeed faster in processing incoming requests, especially for more complex procedures such as Registration and PDU Session Establishment

where improvements up to 50% are achieved. This insight is particularly important because the faster these procedures are completed, the faster the UEs can start sending their data.

From the perspective of operators, PP5GS is considerably easier to deploy because they do not need to consider the inter-NF dependencies and their impact on performance. Moreover, having knowledge of the input traffic, the operators can better orchestrate their networks by deploying and scaling PPNFs accordingly. The operators can choose to offload some PPNFs to the edge and thus provide better performance to the users while reducing the incoming control traffic to the central core.

However, a limitation of PP5GS is that it requires a procedure-aware traffic router able to distinguish and forward the incoming traffic to the correct PPNFs. This entity would then be deployed between gNBs and PPNFs, abstracting the architecture of 5GC. In future generations of mobile networks this routing logic can be integrated directly in the gNB, similarly to procedure-aware routing in our gNBEmu. Moreover, in PP5GS it can be difficult to leverage multi-vendor deployments, at least using the same definition as of today. However, multi-vendor deployments are still viable in a system where each PPNF is developed from a different vendor, as long as the state information is stored remotely in a standardized format.

In the future, we plan to evaluate the data plane performance and assess how adopting PP5GS in the control plane impacts the user experience. Furthermore, we deem it important to investigate the behavior of PP5GS in dynamic autoscaling deployments and in the face of failures to then propose adequate recovery mechanisms.

APPENDIX A
TRAFFIC GENERATION - GNB & UE EMULATOR

Our proposed architecture focuses on improving the performance of the 5GC control plane while omitting data plane evaluations since the functional decomposition of 5GC does not affect the UPF to the same degree as the other NFs. Therefore, benchmarking PP5GS and comparing it with other stateful and stateless deployments requires a tool capable of generating a high volume of control plane input traffic (UE requests). UERANSIM [42] is one open-source tool that allows researchers to deploy multiple UE and gNB instances and evaluate the correctness of their 5GC deployments. However, UERANSIM executes new UEs as separate processes instead of simply emulating their communication, thus making it not suitable for our control plane stress-testing evaluations where hundreds of UEs/s are initialized.

Therefore, in the absence of open-source tools that would satisfy our requirements, we developed a gNB and UE Emulator (*gNBEmu*). This tool supports the parallel execution of UE-related control plane procedures for an arbitrary number of UEs. In its current version, it implements the necessary logic for the emulation of the four procedures mentioned in Table I, and can be easily extended to support more procedures. It is developed in Go language and leverages the open-source libraries of Free5GC such as the ones related to NGAP/NAS communication, MongoDB API, UE authentication, etc.

```
database:
name:    free5gc
url:     mongodb://10.244.0.120:27017

ue:
servPLMNID:      20893
cipheringAlg:    0
integrityAlg:    2

emulation:
gnbIPAddress:    192.168.160.10
gnbSCTPPort:     9487
tasks:
- amf:  192.168.160.1:30320
procedure:       registration
...
firstUEID:       1
warmupNumberOfUEs:       20
totalEntriesInDB:        100000
emulNumberofUEs:         1000
newUEsPerSecond:         25
IATms:   3
```

Listing 1. Sample configuration file for gNBEmu.

Before describing in more details the design and implementation of gNBEmu, we would like to point out a fundamental difference that exists between control plane and data plane traffic generators. Generally, when generating traffic in the data plane, we simply need to make sure that data packets are sent fast and the overall traffic meets the predefined model parameters. This is however not enough for the control plane communication. As explained in Section IV-B, the successful completion of control plane procedures requires exchanging a standardized set of messages between 5G-RAN and 5GC. This means that the traffic generator must be able to initiate new procedures, correctly parse the response coming from AMF and use the received information to build the subsequent messages. Similar to AMF, *gNBEmu* implements a Finite State Machine (FSM) to keep track of the *state* of each UE in order to correctly execute the control plane procedures. Initiating a procedure for a UE is done using an external trigger. After that, with each successful message exchange, the FSM advances to the next state until the execution of the procedure has finished. FSM is a fast mechanism that allows us to automate the execution of procedures and ensure correctness by quickly evaluating the response coming from 5GC.

To better understand the operation of gNBEmu and its capabilities, we summarize its execution flow below:

1) Emulator's context is initialized by parsing the configuration file. As shown in Listing 1, the configuration file contains a number of different parameters that define the behavior of gNBEmu. We divide these parameters in three categories:
   - **`database` -** This block contains the name and URL of the database (DB) that stores UE-related information. It is the same DB instance that is later used by NRF and UDR to retrieve/store information (i.e., the MongoDB instance in case of Free5GC).
   - **`ue` -** Here we provide information for the emulated UEs, such as the Public Land Mobile Network ID (PLMNID) and the ciphering and integrity algorithms used to encrypt/decrypt NAS messages.

- **`emulation` -** This block contains information that defines the emulation process. For example, `emulNumberofUEs` represents the number of UEs to be emulated, `totalEntriesInDB` represents the number of UE entries in the DB and it can be higher than the emulated UEs, `newUEsPerSecond` is the number of new procedures to be triggered every second, while `IATms` specifies the interarrival in milliseconds between each trigger. The type of control plane procedures to be executed together with the AMF's NGAP server information are given in the `tasks` list.

2) In a preemptive manner to avoid any delay during the emulation process, gNBEmu initializes the contexts for all the UEs that will be emulated. Therefore, unique International Mobile Subscriber Identity (IMSI) and authentication keys are generated for each UE. Additionally, a unique `RAN_UE_NGAP_ID` is assigned to each UE for the purpose of identifying it within the gNB.
3) Next, the generated contexts are used to build entries for the DB. These entries are then submitted to the DB in chunks (default chunk size is 20000 entries) which greatly reduces the time to execute this step. *Indexing* based on the UE-ID is also enabled because it is crucial for the optimal operation of the DB instance.
4) gNBEmu enters the control plane procedures execution phase. This phase is actually split into two stages: i) *warmup* where a burst execution for `warmupNumberOfUEs` is performed, and ii) *evaluation* where gNBEmu executes the procedures for `emulNumberofUEs` and measurements are collected. In the *evaluation* stage, the generated traffic follows the parameters set by `newUEsPerSecond` and `IATms`. Both these stages are performed for each of the specified procedures in the configuration file, with a predefined sleep period in between. New UEs are triggered using `goroutines` which are lightweight execution threads. This enables concurrency and avoids issues when spawning the execution of multiple UEs in parallel.
5) Submitting the measurements to the collector application via HTTP communication. During the *evaluation* stage, the beginning and end of all the procedures are timestamped for each emulated UE. Using these values, the Procedure Completion Times are calculated and they are then submitted for post-processing to the collector.

Given that the NGAP/NAS RAN-Core communication runs over Stream Control Transmission Protocol (SCTP), we have made some modifications to its parameters that allow for faster execution and thus more load generated towards 5GC. For instance, after observing the exchanged packet sizes, we set the SCTP socket's `read_buffer_size` to 256 bytes. This modification allows for received messages to be dispatched and processed as fast as possible. Additionally, the default configuration of SCTP enables a Nagle-like algorithm that reduces the number of packets in the network by enqueuing them until a threshold is reached. However, this causes additional delays in the network and therefore we have disabled it. The aforementioned modifications are done also on the 5GC side.

Lastly, we would like to highlight the flexibility that gNBEmu gives us with regards to traffic forwarding. In this work we propose a functional split that is based on control plane procedures, as explained in Section IV-B. Such an architecture means that each of the PPNFs implements an NGAP server and RAN needs to connect to all of them. Therefore, to evaluate PP5GS we need to split the input traffic according to the procedure it belongs to and forward it to the correct endpoints. To achieve this, we implement a procedure-based traffic forwarding mechanism in gNBEmu. For each procedure, we provide the endpoint information in the `tasks` entry of the configuration file, as explained above.

APPENDIX B
UPF EMULATOR

As mentioned in Section III-C, the Free5GC project includes a software implementation of UPF. However, when performing initial tests we noticed that there was a scalability issue with the UPF. First, we tried increasing the buffer sizes in UPF so that they would not overflow during the emulation of high number of UEs. After some further testing, it was observed that this modification was not enough to solve the issue because the UPF would again break once we increase the number of emulated UEs in parallel. Given this situation, we realized that it would be necessary to develop a lightweight UPF Emulator (UPFe) that would allow us to evaluate the PDU Session Establishment and PDU Session Release procedures for the same input rates as Registration and Deregistration. The only requirements for UPFe were to be able to connect to the Packet Forwarding Control Protocol (PFCP) endpoint (e.g., SMF or PPNF), correctly parse the incoming messages, and build the response messages indicating creation/modification/deletion of the data plane resources. Since we focus explicitly on the control plane performance, UPFe does not implement any mechanism to allocate resources to the UEs and initialize data plane tunnels. Similarly to gNBEmu, we develop UPFe in Go and leverage the open-source PFCP library shipped with Free5GC.

REFERENCES

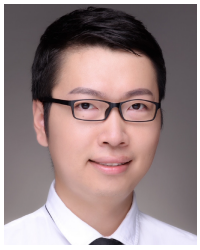
[1] J. G. Andrews et al., "What will 5G be?" *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, Jun. 2014.
[2] I. Widjaja, P. Bosch, and H. La Roche, "Comparison of MME signaling loads for long-term-evolution architectures," in *Proc. IEEE 70th Veh. Technol. Conf.*, 2009, pp. 1–5.
[3] "Signaling is growing 50% faster than data traffic." Nokia. Accessed: Jul. 2022. [Online]. Available: https://bit.ly/3OHKx4R
[4] A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, K. Van der Merwe, and S. Rangarajan, "Scaling the LTE control-plane for future mobile access," in *Proc. ACM CoNEXT*, 2015, p. 19.
[5] *System Architecture for the 5G System (Rel 15)*, 3GPP Standard TS 23.501, 2021.
[6] A. Mohammadkhan, K. K. Ramakrishnan, and V. A. Jain, "CleanG—Improving the architecture and protocols for future cellular networks with NFV," *IEEE/ACM Trans. Netw.*, vol. 28, no. 6, pp. 2559–2572, Dec. 2020.
[7] P. Kiss, A. Reale, C. J. Ferrari, and Z. Istenes, "Deployment of IoT applications on 5G edge," in *Proc. IEEE Int. Conf. Future IoT Technol.*, 2018, pp. 1–9.
[8] Y. Li, Z. Yuan, and C. Peng, "A control-plane perspective on reducing data access latency in LTE networks," in *Proc. ACM MobiCom*, 2017, pp. 56–69.
[9] E. Goshi, M. Jarschel, R. Pries, M. He, and W. Kellerer, "Investigating inter-NF dependencies in cloud-native 5G core networks," in *Proc. CNSM*, 2021, pp. 370–374.

[10] M. Moradi, Y. Lin, Z. M. Mao, S. Sen, and O. Spatscheck, "SoftBox: A customizable, low-latency, and scalable 5G core network architecture," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 438–456, Mar. 2018.
[11] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker, "A high performance packet core for next generation cellular networks," in *Proc. ACM SIGCOMM*, 2017, pp. 348–361.
[12] R. Shah, V. Kumar, M. Vutukuru, and P. Kulkarni, "TurboEPC: Leveraging dataplane programmability to accelerate the mobile packet core," in *Proc. ACM SOSR*, 2020, pp. 83–95.
[13] E. Goshi, R. Stahl, M. He, R. Pries, and W. Kellerer, "Procedure-based functional decomposition for 5G core network functions," in *Proc. KuVS Fachgespräch Netw. Softw.*, 2022. [Online]. Available: https://publikationen.uni-tuebingen.de/xmlui/handle/10900/126089
[14] M. Ahmad et al., "A low latency and consistent cellular control plane," in *Proc. ACM SIGCOMM*, 2020, pp. 648–661.
[15] M. T. Raza, D. Kim, K.-H. Kim, S. Lu, and M. Gerla, "Rethinking LTE network functions virtualization," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, 2017, pp. 1–10.
[16] J. Cho, R. Stutsman, and J. Van der Merwe, "MobileStream: A scalable, programmable and evolvable mobile core control plane platform," in *Proc. ACM CoNEXT*, 2018, pp. 293–306.
[17] M. Corici, E. Troudt, P. Chakraborty, and T. Magedanz, "An ultra-flexible software architecture concept for 6G core networks," in *Proc. IEEE 5GWF*, 2021, pp. 400–405.
[18] 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. ACM SOSR*, 2019, pp. 69–83.
[19] X. An, F. Pianese, I. Widjaja, and U. G. Acer, "dMME: Virtualizing LTE mobility management," in *Proc. IEEE LCN*, 2011, pp. 528–536.
[20] B. Nguyen et al., "ECHO: A reliable distributed cellular core network for hyper-scale public clouds," in *Proc. ACM MobiCom*, 2018, pp. 163–178.
[21] U. Kulkarni, A. Sheoran, and S. Fahmy, "The cost of stateless network functions in 5G," in *Proc. ANCS*, 2021, pp. 73–79.
[22] "Kubernetes." Accessed: Jul. 2022. [Online]. Available: https://kubernetes.io/
[23] *Network Architecture (Rel 14)*, 3GPP Standard TS 23.002, 2017.
[24] *IMT for 2020 and Beyond*, Rec. ITU-R M.2150, Int. Telecommun. Union, Geneva, Switzerland, 2016.
[25] S. Rommer, P. Hedman, M. Olsson, L. Frid, S. Sultana, and C. Mulligan, *5G Core Networks: Powering Digitalization*. London, U.K.: Elsevier Sci., 2019.
[26] "OpenAPI Initiative." OpenAPI. Accessed: Jul. 2022. [Online]. Available: https://www.openapis.org/
[27] "Free5GC." Accessed: Jul. 2022. [Online]. Available: https://www.free5gc.org/
[28] "NextEPC." Accessed: Jul. 2022. [Online]. Available: https://nextepc.org/
[29] K.-L. Lee, C.-N. Lee, and M.-F. Lee, "Realizing 5G network slicing provisioning with open source software," in *Proc. APSIPA ASC*, 2021, pp. 1923–1930.
[30] "Go." Google. Accessed: Jul. 2022. [Online]. Available: https://go.dev/
[31] "MongoDB." Accessed: Jul. 2022. [Online]. Available: https://www.mongodb.com/
[32] "Open5GCore." Fraunhofer FOKUS. Accessed: Jul. 2022. [Online]. Available: https://www.open5gcore.org/
[33] "OpenAirInterface 5GCN." OpenAirInterface. Accessed: Jul. 2022. [Online]. Available: https://openairinterface.org/oai-5g-core-network-project/
[34] "Open5GS." Accessed: Jul. 2022. [Online]. Available: https://open5gs.org/
[35] "Kubernetes documentation." Kubernetes. Accessed: Jul. 2022. [Online]. Available: https://kubernetes.io/docs/concepts/
[36] "BadgerDB." Dgraph. Accessed: Jul. 2022. [Online]. Available: https://github.com/dgraph-io/badger
[37] "Redis." Accessed: Jul. 2022. [Online]. Available: https://redis.io/
[38] Envoy Project Authors, "Envoy proxy." Accessed: Jul. 2022. [Online]. Available: https://www.envoyproxy.io/
[39] Istio Authors, "Istio." Accessed: Jul. 2022. [Online]. Available: https://istio.io/
[40] "Calico." Tigera. Accessed: Jul. 2022. [Online]. Available: https://www.tigera.io/project-calico/
[41] Prometheus Authors, "Prometheus." Accessed: Jul. 2022. [Online]. Available: https://prometheus.io/
[42] A. Güngör, "UERANSIM." Accessed: Jul. 2022. [Online]. Available: https://github.com/aligungr/UERANSIM

Endri Goshi (Graduate Student Member, IEEE) received the Bachelor of Science degree in electronics engineering from the Polytechnic University of Tirana, Albania, in July 2015, and the master's degree in communication engineering from the Technical University of Munich (TUM) in May 2019. He then joined the Chair of Communication Networks, TUM, as a Research and Teaching Associate. His research interests include virtualized and cloud-native mobile core networks, software-defined networks, and programmable networks.

Raffael Stahl received the bachelor's degree in electrical engineering from the Technical University of Munich in April 2020, where he is currently pursuing the master's degree in electrical engineering, specializing in mobile communication network core architectures and embedded systems.

Hasanin Harkous received the master's degree in communications engineering from the Technical University of Munich (TUM) in 2018, and the Ph.D. degree from Nokia Bell Labs in collaboration with the Chair of Communication Networks, TUM. He is a System Architecture Research Engineer with Nokia working on 6G-related topics. His research focuses on topics related to programmable data planes, performance evaluation and modeling, and hardware acceleration in telco cloud technologies.



Mu He (Member, IEEE) received the master's degree in communication engineering and the Doctoral degree from the Technical University of Munich (TUM) in 2015 and 2020, respectively, where he became a Research Associate with the Chair of Communication Networks of Prof. W. Kellerer. His research interests covered network planning, network optimization problems, P4 data plane, and mobile core network optimizations.

Rastin Pries received the master's and Ph.D. degrees in computer science from the University of Wuerzburg, Germany, in 2004 and 2010, respectively. He is a Research Project Manager with Nokia. His current research interests are on applying edge computing to cellular networks as well as on localization and mapping approaches for real-time digital twinning.

Wolfgang Kellerer (Senior Member, IEEE) is a Full Professor with the Technical University of Munich, where he is heading the Chair of Communication Networks, Department of Electrical and Computer Engineering. He was with NTT DOCOMO's European Research Laboratories for over ten years. He currently serves as an Associate Editor for IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and the Area Editor for Network Virtualization for IEEE COMMUNICATIONS SURVEYS AND TUTORIALS.