Evaluating the performance of open source software implementations of the 5G network core

Gabriel Lando, Lucas Augusto Fonseca Schierholt, Mateus Paludo Milesi, and Juliano Araujo Wickboldt
Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Rio Grande do Sul, Brazil
Email: {glando, jwickboldt}@inf.ufrgs.br {lucas.schierholt, mateus.milesi}@ufrgs.br

***Abstract*—Open source implementations of software-based network components have become a viable alternative to deploy and operate 5G networks. Although these implementations provide great flexibility, overall network performance becomes dependent not only on the choice of the software stack, but also on its combination with suitable hardware. In this context, performance testing becomes an essential tool to assess the behavior of these software-based mobile networks under different deployment scenarios. Therefore, the goal of this work is to analyze how the different open source implementations of the 5G network core behave for the execution of procedures at scale. To achieve this goal, we propose and apply performance tests on some of the main open source implementations of the 5G network core. We focus on the evaluation of the performance of two essential procedures: (i) the user equipment registration and session establishment and (ii) the streaming of user data over parallel data plane connections. Among the main results obtained, it was possible to observe that the open source implementation free5GC presents better performance regarding data plane bit rates, while Open5GS shows better stability during the registration process of multiple devices.**

I. Introduction

To attain the necessary improvements for the fifth generation of mobile networks (5G), a reformulation in the network core was necessary, as it is considered the most critical component of the 5G network [1]. The network core (5GC) is responsible for establishing a stable and secure connection between devices and providing access to fundamental services, such as authentication, mobility, and slicing.

Researchers, standard organizations, and industry coalitions have already recognized that softwarization, virtualization, and disaggregation of network functions are key enablers of 5G [2]. This recognition has stimulated open source projects for the implementation of 5G radio access and core networks, such as free5GC, Open5GS, and OpenAirInterface (OAI), which allow complete network stacks to be implemented over commercial-off-the-shelf hardware. Software-based implementations facilitate the evolution of the network, allowing the development and rollout of new functionality without necessarily replacing hardware. For example, open source software projects have become an important driver to the establishment of private 5G network deployments, which are becoming a viable alternative to operate large-scale wireless networks in industrial environments [3].

Software-based networking provides great flexibility, but that comes at a price. Suppose a particular company wants to deploy their own private 5G network with a specific performance target in mind. For example, they might need run an application with a particular latency budget or connect a large number of small devices. Not only they need to choose the most appropriate software stack, but they also need to understand how that piece of software would perform on top of some specific hardware. In this context, software testing becomes an essential tool to assess the behavior of these mobile networks under different scenarios. Compliance and robustness tests, for example, can ensure that the 5GC functions are implemented as specified. In addition, performance tests are necessary to understand how the various network component implementations perform over specific hardware configurations to meet the desired application requirements.

In this work, we aim to answer the following question: how do open source implementations of the 5GC perform at scale for (*i*) execution of registration procedures and (*ii*) streaming of user data over parallel data plane connections? This work proposes and applies performance tests on some of the main open source implementations of the 5GC, as an extension of the *my5G Tester* introduced by Silveira et al. [4]. As contributions, our work pushes the state of the art of performance testing in softwarized networks, especially in the context of the 5GC, bringing relevant discussions to identify limitations and help increase the maturity levels of open source implementations of 5G networks. In addition, our work contributes to the development and open source release of modular and extensible tools to automate tests and monitor experiments over 5GC implementations, which can be used for further investigations.

In the remainder of this paper we describe, in Section II, the theoretical foundations on 5G and software testing, while related work is discussed in Section III. In Section IV, we present the conceptual building blocks of the developed solution for testing the network, collecting, and analyzing data. In Section V, we present an analysis of the collected data, discussing the overall performance metrics and limitations found in the tested 5GC implementations. Finally, in Section VI, we present final remarks, address limitations of our analysis, and suggest perspective for future work.

II. 5G and Software-based Network Testing

A. 5G Networks

The Stand Alone (SA) architecture of the 5G network introduces a new Radio Access Network (RAN), called gNodeB (gNB), and a network core, called 5G Core (5GC), which includes a total of twelve components interconnected through a service layer. Each component has its specific responsibilities for consuming and providing services to and from other elements of the 5G system, through an Application Programming Interface (API) [5].

The three main components are the *Access and Mobility Management Function* (AMF), the *Session Management Function* (SMF) and the *User Plane Function* (UPF). The AMF is responsible for ensuring that the communication process takes place in a cohesive and transparent manner, considering user mobility as the main factor. The SMF is responsible for establishing, modifying, and releasing sessions for each *User Equipment* (UE), in addition to requesting to the UPF the allocation of an IP address to these UEs. The UPF processes and forwards data flowing from and to the UEs and any external network. The remaining nine components of the 5G core perform a variety of functions, such as authentication of UEs, applying security and control policies, control of network slice selection by UEs, integration with non-3GPP access networks, among others.

The *Non-Access Stratum* (NAS) and the *NG Application Protocol* (NGAP) are the two main protocols that implement the control plane communication between the 5GC and the RAN. NAS is used to communicate between the UE and the AMF core function for both 3GPP and non-3GPP devices. The main function of the NAS protocol is to support UE mobility, including procedures such as authentication, identification, and updating UE settings. The NAS is also responsible for supporting session management procedures to establish and maintain the data connection between the UE and the data network [6]. NGAP is the standard protocol for control plane communication between the RAN and the network core, it encapsulates and transports NAS messages, coming from the UE to the AMF. NGAP also manages UE context and *Packet Data Unit* (PDU) sessions [7]. PDU sessions are responsible for providing the end-to-end user plane connection between the UE and the external data network through the UPF. The data stream carried through PDU sessions is called a PDU stream and its data packets are encapsulated over UDP using a GTP-U tunnel [8].

B. Software-based Mobile Network Testing

According to Bertolino [9], software testing consists of dynamically verifying the behavior of a program using set of suitably selected test cases in relation to the specified expected behavior. Regarding the behavior of wireless networks implemented in software, it is important to run tests to ensure that the network works as expected. Among the types of tests that can be performed on top of *open source* 5G network implementations stand out conformance, robustness, and performance tests [4], [10].

Sarikaya [11] defines conformance testing as the activity performed for the purpose of verifying the capabilities and behavior of an implementation under test against the conformance requirements provided in the protocol standard. The use of conformance tests on open source 5G network implementations is important to verify that they behave according to the 3GPP specification.

According to Radatz et al. [12], robustness is defined as the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions. Robustness tests in the core of mobile networks verify the behavior of the network under atypical situations. Silveira et al. [4] divide the group of robustness tests into subgroups, involving registration, authentication and security tests, in addition to specific tests for each protocol and core network function.

Performance tests are important to understand the network behavior under specific workloads. For example, a performance test could be designed to simulate a network usage overload, which can cause fluctuations in metrics, such as throughput, packet loss, and jitter, or event cause the network to malfunction partially or completely. For the execution of performance tests, it is customary to use reference applications, called benchmarks, which reproduce the same input parameters to comparatively evaluate the performance of different systems or services [13]. Metrics used in performance tests of software-based mobile networks can be divided into two groups [14]: (*i*) communication related metrics, such as data transfer rates, latency, and packet loss, and (*ii*) resource consumption related metrics, such as processor load, memory usage, and execution time.

III. Related Work

Silveira et al. [4] introduce a proof of concept 5GC tester, focusing on conformance and robustness tests considering three of the most preeminent open source implementations available today: *free5GC*, *Open5GS* and *OAI*. The study concluded that all tested 5GCs had satisfactory results under normal conditions (i.e., they comply to the specification). However, among the three implementations analyzed, *OAI* was considered by the authors as the least mature.

The tests performed by Silveira et al. [4] are extremely important to evaluate the correct behavior of a 5G network core. As briefly described in the article, the proof-of-concept implementation also supports performing tests where multiple UEs sequentially connect to the network to assess whether the implementation of the 5GC supports repeated executions of standard procedures. However, tests implemented by Silveira et al. did not intend to compare the performance of these implementations in cases where a large number of UEs connect to the core in parallel, for instance, making registration and authentication requests in short periods of time. This type of test would be important to evaluate the performance of implementations under heavy workloads.

Based on the free5GC implementation, Lee et al. [14] deals with 5G network slicing, which allows the creation of multiple logical networks on top of a single physical substrate. The authors performed experiments to verify the conformance of the network slicing functionality, as well as its performance. They measured the performance of the network using specific applications to overload the data network. It was found that latency gradually increased with the number of connections, and packets were lost at a certain point due to network overload. The experiments performed by Lee et al. [14] are very important to analyze how the overload of mobile networks affects the performance of sliced 5G networks. However, this article carried out the experiments on the user plane of the network, not taking into account how much the control plane of the network was affected.

The study by Garcia et al. [15] conducts performance tests on the user plane of real mobile networks of past generations to compare the performance LTE in relation to the 3.5G network using different scenarios and operators. The metrics collected by the study were latency, throughput, and the loading time of web pages. Each test was performed with and without background traffic. This study only analyzes data traffic on mobile networks, verifying the behavior of networks in normal use and with background traffic. Since experiments were carried out on real networks of four local operators, it is understood that the study could not take measures related to the control plane performance.

IV. Performance Tester for 5GC

In this paper, we propose and apply performance tests in open source 5GC implementations as an extension of the work of Silveira et al. [4], which introduces a proof of concept testing framework called *my5G Tester*. Silveira et al. proposed and implemented a modular and extensible architecture mainly focused in the design of conformance and robustness tests on different open source 5GC implementations. Since *my5G Tester* supports expansion to other workloads, allowing the development of other types of tests, we have proposed, implemented, and integrated extra modules to allow performance tests to be carried out[1].

The high-level architecture presented in Figure 1 highlights some of the main components we have adapted from the *my5G Tester* framework, which are divided into three main layers: Simulation, Controller and User Interface. The Simulation layer mimics the behavior of the RAN, allowing the creation of virtual gNB and UE devices capable of connecting and exchanging control messages and data traffic with the 5GC. In this work, since we intend to simulate heavy workloads, we introduce in the Simulation layer the possibility to connect multiple simultaneous UEs to test 5GC implementations with many parallel execution of different procedures. The Controller layer receives user commands through an interface, selects tests to be executed from a set of Test plans, interacts with the Simulation layer configuring gNBs and UEs, and sends experiment metrics to the Metric Collector, which will store them for future use. The present work extends the Test plan module, adding the possibility to run connectivity tests of UEs in parallel, to evaluate the operation of 5GC implementations under heavy workloads. Finally, the User Interface layer allows execution and monitoring of tests, as well as exporting metrics

[1]Source codes are available at the PORVIR-5G project GitHub repository: https://github.com/PORVIR-5G-Project

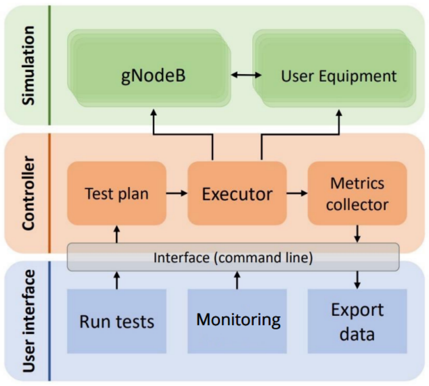


Fig. 1. Tester Architecture

stored in the Controller layer. Within this layer the user can interact with the tester through parameters sent by a command line to be executed in the test environment.

Differently from conformance or robustness tests, which are expected to return always the same result (true or false values) for every run of a test plan, performance test results might fluctuate with the randomness introduced by concurrent access to resources. Therefore, in addition to the modules implemented internally in the tester, it was also necessary to design and implement mechanisms for automating the creation and configuration of the test environment, as well as the execution of tests. The internal modules and mechanisms implemented to orchestrate multiple runs of test plans as experiments and collect the generated data for further analysis are detailed in the following.

A. Test Module

For the execution of performance tests, a module was developed for the tester that allows the execution of multiple connections of simultaneous UEs. This module was implemented supporting configuration parameters to facilitate variations between tests. Therefore, it is possible to define the number of UEs to be connected, the delay in milliseconds (ms) between one connection and another and the delay in seconds (s) to start the experiment execution.

At the beginning of execution, this module simulates a gNB and starts connecting it to the 5GC. After the gNB connection is successful, the tester waits for a configurable amount of time to start running the experiment. After that the module starts a parallel workflow to create a new UE instance and connect it to the 5GC. After starting the parallel workflow, the module waits for the user-defined interconnect delay before starting the next workflow, until the defined number of devices is reached.

During the module development, a limitation was found in the implementation of the *my5G Tester* that did not allow the simulation of more than 255 UEs connected simultaneously. Analyzing the code and discussing with original developers, we observed that the correction of the limitation would require time and resources unanticipated for the scope of this project. Therefore, instead of fixing the implementation, to circumvent this limitation, we decided to use multiple instances of the tester. Thus, each instance simulates a different gNB connected simultaneously to the 5GC and, to each gNB, we connect up to 255 UEs, allowing the simulation of large scale a RAN.

B. *Data Collection Module*

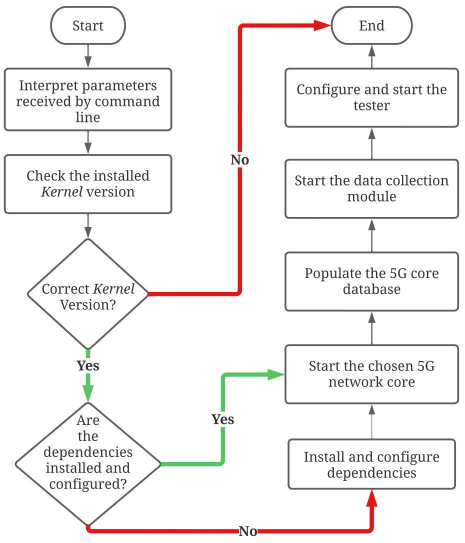
The data collection module is responsible for gathering data both from the inner workings of the 5GC components as well as from the infrastructure that surrounds it. Regarding the internal 5GC components behavior, our data collection module runs a parallel execution flow that records the time, in nanoseconds, that each UE took to switch between states during its connection to the 5GC (e.g., registration initialization and acceptance, PDU session establishment, etc.). The timing information for each state change is written to the standard text output of the tester instance for later collection and analysis. This module provides valuable information for the breakdown of the overall execution time of the registration procedure, providing insight on the contribution of each individual state change in relation to the whole process.

Regarding infrastructure metrics, this module configures monitoring flows using an application called *Node-RED*. Since we use Docker to deploy the 5GC components (more details on the experimentation setup in Section V), a flow was developed to list all containers running on a host machine, collect metrics from the *Docker* stats API, and store them in a time series database called *InfluxDB*. This flow was set to run automatically after *Node-RED* is initialized, allowing the collection of disk, network, processor, and memory usage, among other metrics, in an automated way.

C. *Orchestrator*

In order to carry out all the executions of the experiments as similarly as possible, an orchestrator was developed that manages the entire execution of the experiment. Figure 2 represents the orchestrator's execution algorithm. The orchestrator runs through a command-line user interface, supporting several parameters to customize and automate the experiment execution in a simplified way. The orchestrator is designed to support different implementations of the 5GC.

When starting the orchestrator, an analysis of the compatibility of the operating system's kernel is carried out, verifying if it meets the requirements of the 5GC. Afterward, the orchestrator verifies that all dependencies are correctly installed and configured. Otherwise, the it makes the necessary modifications automatically. Once the system is ready, the execution of the desired experiment begins. First, the chosen 5GC source code is downloaded, compiled, and built into *Docker* images, which are then launched. Then, the core's database is filled with the necessary information for connecting the UEs according to the experiment settings. Finally, the data collection module and the tester, with the user-defined parameters, are started as containers.



Start
Interpret parameters received by command line
Check the installed *Kernel* version
Correct *Kernel* Version?
Yes
No
Are the dependencies installed and configured?
Yes
No
Install and configure dependencies
Start the chosen 5G network core
Populate the 5G core database
Start the data collection module
Configure and start the tester
End

Fig. 2. Orchestrator Execution Flowchart

V. Analysis of Collected Data

A. *Infrastructure and parameters*

To carry out the experiments and validate the results we used virtual machines running the Ubuntu Server 20.04.4 LTS operating system with different resource configurations: 4, 6, 8 and 12 virtual cores of an *Intel Core i7* processor, 4 and 8 GB RAM, and a 64 GB virtual disk stored on an M.2 NVMe SSD. Summary details of the setup can be found in Table I.

TABLE I
Test Machine Configuration

	Virtual Machines			
CPU	Intel Core i7 8700T @ 2.4 GHz			
	4 cores	6 cores	8 cores	12 cores
RAM	DDR4 2666 MHz			
	4 GB	8 GB	8 GB	8 GB
Storage	SSD M.2 64GB NVMe			
Operating System	Ubuntu Server 20.04.4 LTS			

Two different experiments were carried out on different configurations of this infrastructure. The experiments were 5GC session establishment and registration times tests and 5GC data plane performance tests.

B. *Session Establishment and Registration Time Tests*

The experiment for measuring registration times and session establishment of the 5G core consists of simulating multiple UEs trying to connect and establish a PDU session with the 5G core to be tested. In this way, it is possible to measure the latency between each stage of the connection and its variation according to the workload applied to the core. In order to be able to measure this latency in different scenarios in an automated way, configuration parameters were added both in the experiment orchestrator and in the testing module. In this experiment, the metrics of processor, RAM, disk and network usage were also collected from all components of the 5GC and the tester, using the tools for exporting metrics from the *Docker* containers.

For this experiment, a virtual machine configuration with 12 virtual cores and 8 GB RAM was used. 1Regarding the experiment settings, the 5GC versions *free5GC* v3.2.1 were used, the most recent being available at the time of the experiments, and *Open5GS* v2.3.6, which is the version recommended by the tester developers. For each core, the following tester settings were used: number of gNB ranging from 1 to 11, with a step of 2, interval between connections from 100 ms to 500 ms, with a step of 100 ms, and each gNB connecting 100 UEs. This experiment was performed 10 times, to obtain an amount of relevant data, where external noises to the experiment were mitigated.

During the tests to carry out, some limitations were found and should be highlighted. Firstly, we intended to use *free5GC* version v3.0.6, which is recommended by the developers of the tester. However, this version failed after connecting 10 UEs, which would make this experiment unfeasible. OAI v1.4.0, which is the latest available, had a similar limitation failing after connecting exactly 15 UEs. *Open5GS* v2.3.6 is able to connect exactly 1024 devices simultaneously before crashing the AMF network function. Therefore, it was decided that at most 11 gNBs would used in our experiments, accounting for a total of 1100 connected UEs.

C. Data Plane Performance Tests

The experiment for measuring the performance of the 5G core data plane consists of simulating 1 or more UEs performing the registration in the network core and initiating a data plane. After the registration is completed, the data plane capacity is measured using *iPerf* version 2.0.

To carry out this experiment, it was decided to use version 2.0 of *iPerf*, instead of the latest version, due to its ability to support multiple client connections on a single server instance, in addition to allowing exporting metrics from the experiment in a comma-separated text file, making it easier to post-process the data. Tests were carried out with all virtual machine configurations of Table I. In this experiment, the main metric to be observed is the bit rate, in bits per second (Bps), representing the maximum data traffic capacity between the UE and the 5G network, passing through the UPF. In this experiment, the metrics of processor, RAM, disk and network usage were also collected from all the components of the 5GC core and the tester, using the tools for exporting metrics from the *Docker* containers.

Regarding the parameters used for the execution of the experiment, the configuration of 1, 2, 4, 6, 8 and 10 gNBs with 1 UE each was used, recording and connecting the data plane before the execution of each performance test. As for the parameters of the *iPerf* tool, the experiment execution time was set to 60 s, reporting the test metrics every second. The tested 5GCs were *free5GC* v3.0.6 and *Open5GS* v2.3.6, both recommended by the tester. This experiment was performed 16 times, to obtain an amount of relevant data, where external noises to the experiment were attenuated.

Some limitations were observed during the preparation and execution of this experiment. Since the latest version of *free5GC* was used in the previous experiment, it was intended to use the same version for this experiment. However, the tutorials for the v3.2.1 implementation failed to establish the necessary routes for the data plane, making the connection between the *iPerf* server and client unfeasible. Therefore, we decided to use a previous version of this 5GC, which had been approved by the tester's developers. In this experiment, the OAI had the same problem, with failures to establish the routes between the UE and the 5G network in versions v1.3.0 and v1.4.0, the latter being the most recent to date. Therefore, it was decided not to carry out the experiments on this 5GC.

D. Discussion on Session Establishment and Registration

For the experiment to measure the connection time of UEs, the total time between the beginning of the registration process and obtaining the data plane was taken into account. After collecting and processing the data from this experiment, the charts in Figure 3 were generated to allow the visualization and analysis of the collected data. The chart represents the average connection time for each burst of simultaneous connections, with the X axis representing the experiment execution time, in s, and the Y axis representing the total time to establish the connection and start the data plane of this UE, in ms. In the left column, the results for *free5GC* are presented, and in the right, *Open5GS*. For each 5GC, 5 charts were drawn, with different variations in the interval between bursts of UEs connections.

When analyzing the data, it is possible to infer that, for the conditions of the experiment, *free5GC* had an average connection time of 1089 ms, when each connection started 500 ms after the previous one. In this experiment, the median connection time was 853 ms and the maximum connection time was 5.57 s. For an interval between connections of 100 ms, the average connection time was 7731 ms, with its median being 3027 ms and the maximum just over 54 s.

As for the behavior of *Open5GS*, it can be seen that for an interval of 500 ms, the average time was 306 ms, with its median at 305 ms and its maximum connection time at 345 ms. For the 100 ms interval, the mean rose to 364 ms, while the median rose to 358 ms and the maximum value reached 490 ms. Thus, considering only the connection time, *Open5GS* shows better performance, withstanding several bursts of connections in different scenarios, with its average time varying around 58 ms.

During the execution of the experiments, the rate of unsuccessful connections between the UE and the core of the network was significant between each execution of the experiment for each of the cores tested. Therefore, the charts in Figure 4 were drawn with the comparison of the average error rate between the two evaluated implementations.

In the boxplot, the line inside the box represents the median of the connection error rate values. The ends represent half of

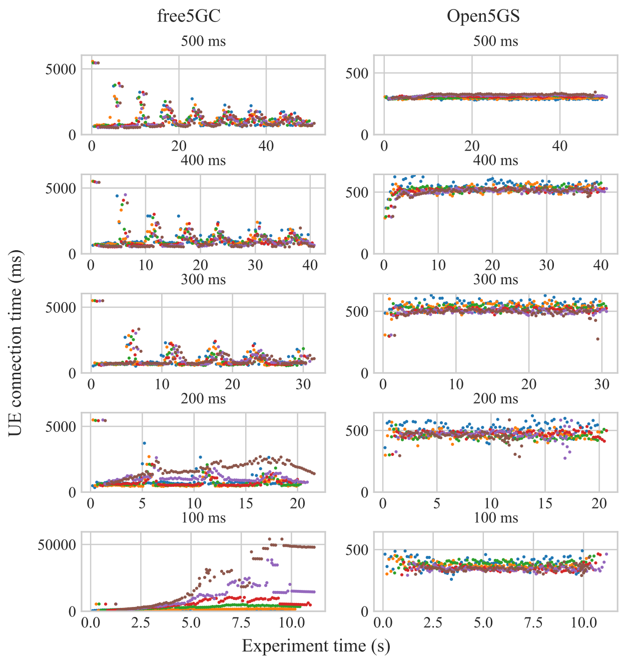


Fig. 3. PDU session registration and establishment time for each 5G core

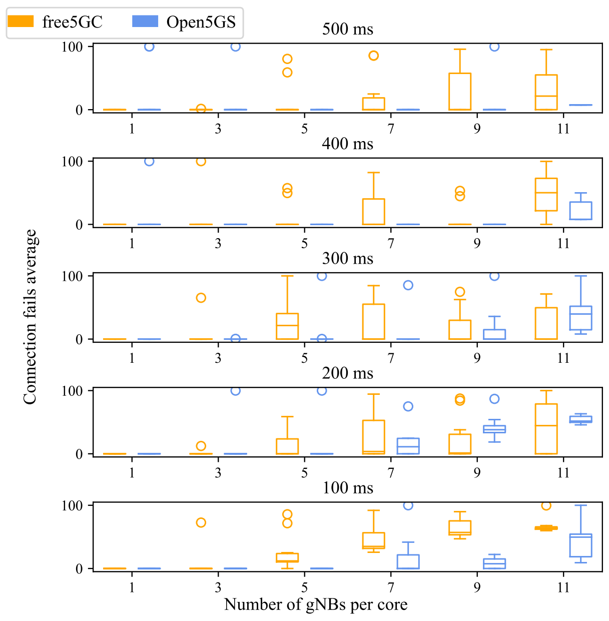


Fig. 4. Mean and standard deviation of connectivity errors

these values closest to the median. The top and bottom lines represent the extremes of the connection error rates. The small circles represent the outliers. These circles show unrepresentative data for the experiment, which may be caused by external agents during any of the executions of the experiment. The larger the chart box, the larger the data spacing.

In this chart, it is possible to notice that the two cores tend to behave satisfactorily for a low UE concurrent connection rate. However, a few outliers exhibited some failures occurred during the execution. When increasing the UE concurrent connection rate, both cores present instability, having a high variation rate during the execution of the experiment. *Open5GS* presents a consistent failure rate at 11 gNBs. This failures can be explained by the limitation on this implementation. Upon reaching the limit of 1024 devices simultaneously connected, all attempts are unsuccessful.

E. Discussion on Data Plane Performance

For the experiment to evaluate the performance of the data plane of the 5GC, *iPerf* 2.0 was run simultaneously for 60 s on each UE connected to the 5G core. *iPerf* was configured to generate TCP data. In a real world usage, this type of data traffic simulates a file download, for example, when the UE whats to download the file as fast as possible. At first, the virtual machine was used in its maximum configuration, with 12 virtual CPU cores and 8 GB RAM. After collecting and processing the data provided by the *iPerf* tool, charts were generated representing the average bit rate per second of the tested 5GCs.

Figure 5 shows the aggregated value of the bit rate for each UE for all virtual machines. The first line of charts represents the virtual machine configuration with 12 virtual cores and 8 GB RAM. While the last line of charts represents the virtual machine configuration with 4 virtual cores and 4 GB RAM. The X axis of the chart represents the amount of UEs in each round of the experiment, where the bar segment represents the UE of the same color in the two previous figures. Each bar of the chart represents the average bit rate in Mbps aggregated between the UEs.

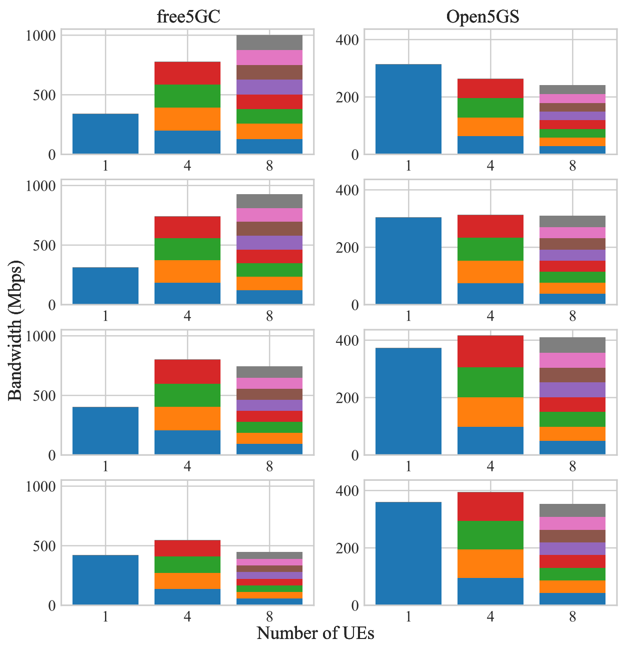


Fig. 5. Bit rate aggregate value for each UE number setting

In this experiment, for the virtual machine with 12 virtual cores and 8 GB RAM, we noticed that in *free5GC* the total bit rate between each execution increases according to the number of connected UEs. The average bit rate value for this core with one UE connected is 338.4 Mbps, increasing to 777.0 Mbps with 4 UEs connected and 1001.1 Mbps with 8 UEs connected. In contrast, the behavior of the *Open5GS* core is opposed to *free5GC*. In the *Open5GS* core, the average bit rate value for a connected UE is 314.5 Mbps, reducing to 262.7 Mbps with 4 UEs connected and 241.1 Mbps when connecting 8 simultaneous UEs. This experiment demonstrates that *free5GC* performs better handling data traffic of UEs at scale than *Open5GS*.

By limiting the amount of resources available for the execution of the experiment, decreasing the amount of virtual processor cores and RAM of the virtual machine, it can be seen that the data plane performance for the *free5GC* core is directly proportional to the number of virtual cores available. In the virtual machine with 8 virtual processor cores and 8 GB RAM, represented by the second line of charts on Figure 5, maximum data plane bit rate for core *free5GC* is reduced to 927.6 Mbps when running with 8 concurrent UEs. This performance reduction can be explained due to the limitation on the maximum processing load supported by the virtual machine.

The performance degradation of the data plane becomes more visible by further limiting the amount of virtual cores available for the experiment. At the third line of charts on the Figure 5, where the virtual machine runs with 6 virtual processor cores and 8 GB RAM, one can see that the maximum data plane performance of *free5GC* is achieved with four simultaneous UEs. The same behavior is present when the virtual machine has its available resources reduced to 4 virtual processor cores and 4 GB RAM. The performance reduction for tests with a greater number of simultaneous UEs is explained by the processor usage overhead caused by the number of UEs generating data traffic in parallel.

F. Discussion on Resource Consumption

This section focuses the analysis of the processor usage during the execution of the experiments. We also collected data on RAM, disk, and network usage, but since these metrics did not provide significant insight to compare both tested implementations, we decided not to present them in this paper. In Figures 6 and 8, the CPU usage data for each container has been grouped into three sets to make it easier to visualize. The first set represents the aggregated data from all instances of the tester, referred to in the charts as Tester. The second set represents the aggregated data of all core functions under test, referred to in the charts as Core. The third set represents the aggregated data from the rest of the running services, such as the containers used to run the core data collection module and the *iPerf* application server used in the second experiment, named in the charts as Others. A more detailed analysis of the use of resources for each component running during the period of the experiment is possible when using the tool to generate views of the collected data available through the *InfluxDB* application. This tool is part of the data collection module and collects more metrics than those presented here.

1) Analysis of resource consumption in the 5GC session establishment and registration tests: For the analysis of the use of processor resources for the execution of the experiment of registration and establishment of the 5G core session, an execution of the experiment was used with an interval between connections of UEs of 500 ms and 9 gNBs connecting UEs in parallel. The aggregated results can be seen in Figure 6.

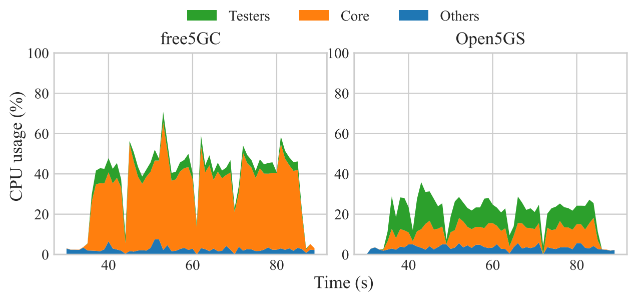


Fig. 6. CPU usage for execution with 500 ms delay between connections

This figure shows the CPU usage according to the experiment execution time for *free5GC* and *Open5GS*. The X axis of the charts represents the time in seconds referring to the total data collection of the experiment, considering the time for the initialization, execution and termination of the experiment, and the Y axis represents the percentage of CPU usage. When looking at resource usage only during the experiment execution period, the average processor usage of *free5GC* was 37.31%, while *Open5GS* used 8.98%. The results described above demonstrate that *Open5GS* presented a 4.1x higher performance in terms of processor usage compared to *free5GC*.

Figure 7 presents the breakdown of processor usage for each component of *free5GC* and *Open5GS* for the session establishment and registration experiment with an interval of 500ms between each burst of connections. As expected for this experiment, the AMF is one of the components that uses most resources in both 5GC implementations. Nevertheless, it is possible to notice that processor usage of other core functions vary for each of the implementation. As an example, while *free5GC* uses a significant amount processing resources in the *Network Repository Function (NRF)*, *Open5GS* uses proportionally more compute resources in Unified data management (UDM) and Policy Control Function (PCF) components. It is important to notice that the *Binding Support Function (BSF)* is not present in this *free5GC* deployment.

2) Analysis of resource consumption in the data plane performance tests: To perform the analysis of the use of resources during the execution of the experiment for the evaluation of the performance of the data plane, the metrics for processor usage resources was chosen. This analysis complements the explanation of the experiment demonstrated in Section V-E. The experiments were conducted in four different

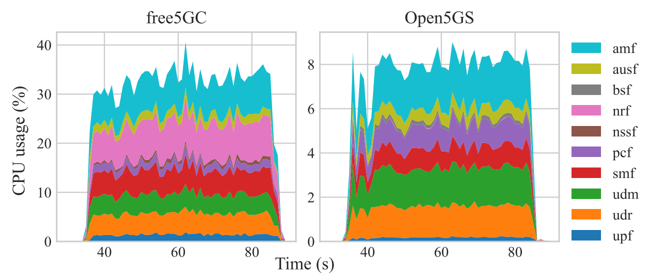


Fig. 7. CPU usage per core component for execution with 500 ms delay between connections

configurations. However, the analysis of resource consumption was focused on only one configuration, using the virtual machine with the lowest amount of resources. By looking at the CPU resource usage for the virtual machine with 4 virtual processor cores and 4GB RAM, it is possible to explain the performance drop of *free5GC* data plane after tests with more than 4 simultaneous UEs. Figure 8 represents the charts of processor usage for an execution of the data plane performance experiment.

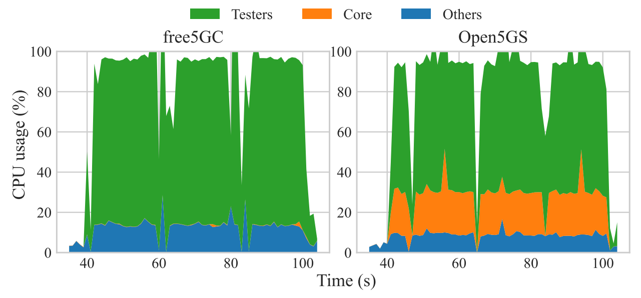


Fig. 8. CPU usage for the data plane experiment with 4 UEs connected

It is possible to observe that the processor usage for the execution of four connected UEs and using the data plane of *free5GC* is very close to 100%, reaching the maximum usage of the processor in several moments. Of the total processor usage, 91.87% of usage is represented by the testers and UEs performing the tests, 0.17% was used by *free5GC*, and the remaining was used for other containers running on this machine. This demonstrates that this 5GC uses few processor resources to manage the data plane of the connected UEs.

Regarding *Open5GS*, which is not optimized to manage the data plane in parallel threads, the total average processor usage during the execution of the experiment was 93.75%. The total processor usage during experiment execution is mainly split between the tester instances and the 5GC, with the tester representing 60.24% of the processor usage and the core representing 22.1%. The reduction in processor usage by the tester instances and UEs compared to the experiment performed with the *free5GC* core is explained by the low bit rates achieved by *Open5GS*.

A possible explanation of these results is that the core *free5GC* probably uses parallelization to manage the data plane processing of the UEs. This would justify the ease of scaling the number of connected UEs, subtly reducing the performance of individual UEs. On the other hand, the *Open5GS* core does not have the same behavior. According to the results, the *Open5GS* core used 22.1% of the total processing available in the virtual machine which is equivalent to slightly less than the capacity of one available processor core. Possibly, this 5G core is not optimized to parallelize data plane processing when multiple UEs are connected. This would explain the poor performance observed in the *Open5GS* core results.

VI. Final Considerations and Future Work

The present work proposed, implemented, and applied performance tests on some of the main open source 5GC implementations. The findings and limitations discussed can be useful to guide the evolution of software-based 5G technologies. Likewise, our proposal can be used to test and evaluate other 5GC implementations.

One of our main contributions to state of the art of performance testing in softwarized networks is relative to the assessment of the maturity level of current open source implementations of the 5GC. The two implementations evaluated have presented major limitations in terms of performance at scale, hindering their implementation in real large-scale 5G network deployments. Our results demonstrated that *free5GC* presents better performance regarding data plane bit rates, scaling up to roughly 1Gbps with multiple UEs connected. On the other hand, *Open5GS* shows better stability during the registration process of multiple UEs, keeping connection times generally under half of a second. Also, this work shows that the *OAI* 5GC implementation was considered least mature and was not able to run any of the performance tests proposed.

Another practical contribution was the development a module integrated in the *my5G Tester*. This module is available in the *PORVIR-5G* project code repository and is open to the community. Thus, it is possible to use the available tester implementation to reproduce the experiments carried out in this work, as well as perform tests on other 5GC implementations.

The present work opens opportunities for future investigations regarding the performance of the 5GC. As an example, one could extend the proposed test mechanisms to evaluate other 5GC procedures, such as handling user mobility, handover, slice selection, etc. Moreover, the results of performance tests carried out could provide meaningful inputs in the form of datasets to help designing intelligent orchestration mechanisms for the 5GC. Based on the performance test results an algorithm could automate decisions upon replication of overly stressed core components, vertically or horizontally scaling the underlying infrastructure, just to name a few.

Acknowledgment

This study was partially funded by CAPES, Brazil - Finance Code 001. This work was also supported by the São Paulo Research Foundation (FAPESP), Brazil grant 2020/05182-3.

References

[1] K. V. Cardoso, C. B. Both, L. R. Prade, C. J. A. Macedo, and V. H. L. Lopes, "A softwarized perspective of the 5g networks," Jun 2020, IEEE NetSoft 2020 - Tutorial. [Online]. Available: http://arxiv.org/abs/2006.10409

[2] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, programmable, and virtualized 5g networks: State-of-the-art and the road ahead," *Computer Networks*, vol. 182, p. 107516, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620311786

[3] A. Aijaz, "Private 5g: The future of industrial wireless," *IEEE Industrial Electronics Magazine*, vol. 14, no. 4, pp. 136–145, 2020.

[4] L. B. Silveira, H. C. de Resende, C. B. Both, J. M. Marquez-Barja, B. Silvestre, and K. V. Cardoso, "Tutorial on communication between access networks and the 5g core," *Computer Networks*, vol. 216, p. 109301, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622003528

[5] 3GPP, "Technical Specifications and Technical Reports for a 5G based 3GPP system," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 21.205, 7 2020, version 16.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/21205.htm

[6] ——, "Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 24.501, 3 2022, version 17.6.0. [Online]. Available: http://www.3gpp.org/DynaReport/24501.htm

[7] ——, "NG-RAN; NG Application Protocol (NGAP)," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.413, 12 2021, version 16.8.0. [Online]. Available: http://www.3gpp.org/DynaReport/38413.htm

[8] ——, "NG-RAN; PDU session user plane protocol," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.415, 12 2021, version 16.6.0. [Online]. Available: http://www.3gpp.org/DynaReport/38415.htm

[9] A. Bertolino, "Software testing research and practice," in *Abstract State Machines*, E. Börger, A. Gargantini, and E. Riccobene, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–21.

[10] P. Zhang, X. Yang, J. Chen, and Y. zhen Huang, "A survey of testing for 5g: Solutions, opportunities, and challenges," *China Communications*, vol. 16, pp. 69–85, 2019.

[11] B. Sarikaya, "Conformance testing: Architectures and test sequences," *Computer Networks and ISDN Systems*, vol. 17, pp. 111–126, 7 1989. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0169755289900044

[12] J. Radatz, A. Geraci, and F. Katki, "IEEE standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, 1990.

[13] C. A. Boano, S. Duquennoy, A. Forster, O. Gnawali, R. Jacob, H.-S. Kim, O. Landsiedel, R. Marfievici, L. Mottola, G. P. Picco, X. Vilajosana, T. Watteyne, and M. Zimmerling, "Iotbench: Towards a benchmark for low-power wireless networking," in *IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench)*. IEEE, 4 2018, pp. 36–41. [Online]. Available: https://ieeexplore.ieee.org/document/8429500/

[14] K. L. Lee, C. N. Lee, and M. F. Lee, "Realizing 5G network slicing provisioning with open source software," in *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2021, pp. 1923–1930.

[15] J. Garcia, S. Alfredsson, and A. Brunstrom, "Delay metrics and delay characteristics: A study of four swedish hsdpa+ and lte networks," in *European Conference on Networks and Communications (EuCNC)*. IEEE, 6 2015, pp. 234–238. [Online]. Available: http://ieeexplore.ieee.org/document/7194075/