# CtrlBench: Towards Accurate API-Level Benchmarking in 5G Core Control Plane

Chanuk Park
*Department of AI-Based Convergence*
*Dankook University*
cupark@dankook.ac.kr

Jaehyun Nam*
*Department of Computer Engineering*
*Dankook University*
namjh@dankook.ac.kr

***Abstract*—The service-based architecture (SBA) of the 5G Core enables cloud-native deployment but makes performance evaluation harder. Existing end-to-end (E2E) metrics, such as Registration Time, compress entire procedures into a single number. This hides which Network Function (NF) or API actually causes delays, and prevents targeted optimization. General HTTP benchmarking tools are also insufficient because they cannot capture realistic 5G-specific workflows such as chained APIs or cryptographic authentication steps. We present *CtrlBench*, a domain-specific microbenchmarking tool for the 5G Core Control Plane. *CtrlBench* parses 3GPP OpenAPI specifications, discovers NF endpoints via the NRF, and automates 5G-AKA authentication with the Milenage algorithm. It can generate configurable load and chain-dependent APIs, giving granular, fine-grained, and reproducible latency measurements. In experiments with Free5GC, *CtrlBench* revealed API-level bottlenecks that conventional E2E metrics could not precisely show. Unlike generic HTTP load generators, it executes the full standardized procedure, ensuring measurements that reflect realistic Control Plane execution rather than synthetic traffic.**

***Index Terms*—5G, Control Plane, API, Microbenchmark**

## I. Introduction

Starting from 3GPP Release 15, the 5G Core was standardized with the Service-Based Architecture (SBA). In this model, each Network Function (NF) interacts via HTTP/2-based APIs, primarily following a RESTful style, while also supporting subscribe/notify patterns for state-change notifications [1], [2]. This cloud-native, microservices-based design replaces the traditional hardware-centric architecture, enabling rapid service deployment, horizontal scalability, and independent NF evolution. Such flexibility underpins the three key 5G service scenarios: enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communications (URLLC), and massive Machine-Type Communications (mMTC) [3]. However, the shift to API-centric operation introduces complexity for performance evaluation, particularly in the Control Plane.

The Control Plane coordinates functions such as authentication, session management, and policy enforcement through multi-step API interactions among NFs including AMF, SMF, AUSF, UDM, and NRF [4]. Even a few milliseconds of delay at each NF accumulate across multiple APIs, degrading end-to-end (E2E) service quality. Existing evaluations, as defined in 3GPP TS 28.552 [5], rely on procedure-level E2E KPIs such as Registration Time and PDU Session Establishment Time. While these metrics indicate overall service quality, they fail to reveal the origin of latency.

A closer look at current evaluation approaches reveals several challenges. First, the granularity of E2E KPIs is too coarse to isolate latency variations at the NF or API level, making it difficult to pinpoint the true source of bottlenecks. Second, the localized effects of NF-level optimizations often remain hidden, since improvements applied to one NF may not be reflected in the overall procedure time. Third, E2E testing environments are heavyweight and entangled, requiring full UE/RAN simulators and complete Core deployments; this coupling of RAN and Core processing complicates reproducibility and prevents consistent measurement of Control Plane performance in isolation. Finally, generic HTTP benchmarking tools cannot faithfully reproduce SBA-specific workflows, since they lack support for dynamic NF discovery, sequential API chaining, and cryptographic operations such as 5G-AKA authentication. Together, these limitations highlight the absence of a domain-specific method for fine-grained, reproducible evaluation of the 5G Core Control Plane.

To address these challenges, we present *CtrlBench*, an API-level microbenchmark tool explicitly designed for the 5G Core Control Plane. Unlike conventional E2E KPIs that collapse complex procedures into a single value, *CtrlBench* enables fine-grained and granular attribution of latency to specific APIs and NFs, providing actionable insights for both system operators and researchers. It automatically parses 3GPP OpenAPI specifications to extract NF-specific APIs and schemas, dynamically discovers endpoints via the NRF, and integrates the Milenage algorithm to reproduce 5G-AKA authentication in a fully automated manner. Beyond static benchmarking, it supports configurable load generation and declarative API chaining, allowing realistic and scalable emulation of SBA workflows under diverse conditions. These capabilities not only expose bottlenecks that remain invisible to aggregated metrics but also deliver reproducible and comprehensive performance baselines that facilitate NF-level optimization, regression testing, and fair comparison across different 5G Core implementations. By bridging the gap between coarse-grained E2E evaluation and domain-specific microbenchmarking, *CtrlBench* establishes a new methodology for systematic and precise analysis of 5G Core Control Plane performance.

* Jaehyun Nam (namjh@dankook.ac.kr) is the corresponding author.

**Contributions.** This paper makes the following contributions:

- *API-level benchmarking framework:* Provides fine-grained measurement and control of prerequisite API execution, enabling quantitative comparison between session reuse and new session creation.
- *5G-specific workflow automation:* Integrates the Milenage algorithm to compute RES* values in real time and propagate them across chained APIs, faithfully reproducing 5G-AKA authentication.
- *OpenAPI-driven testbed simplification:* Combines schema extraction from 3GPP OpenAPI, NRF-based dynamic discovery, and HTTP/2 support to reduce the complexity of Control Plane benchmarking.

**Paper Organization.** The remainder of this paper is structured as follows. Section II provides background and motivation. Section III presents the design of *CtrlBench*, and Section IV details its implementation. Section V evaluates its performance, and Section VI concludes.

## II. Background and Motivation

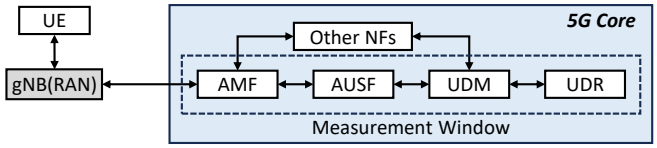
### A. Service-Based Architecture and Control Plane Performance

The 5G Core, standardized in 3GPP Release 15, adopts a SBA where each NF is implemented as a microservice and communicates via HTTP/2-based Service-Based Interfaces (SBIs). Unlike traditional monolithic cores, SBA enables containerized deployment, elastic scaling, and rapid evolution of individual NFs. These features make SBA well-suited for eMBB, URLLC, and mMTC use cases, but they also shift performance evaluation toward distributed API-level interactions. In the Control Plane, procedures such as registration or PDU session establishment are no longer single-step operations, but multi-hop API chains discovered dynamically through the NRF. As a result, latency accumulates across APIs, and subtle NF-level delays can propagate into noticeable E2E performance degradation.

### B. Performance Metrics in Previous Studies

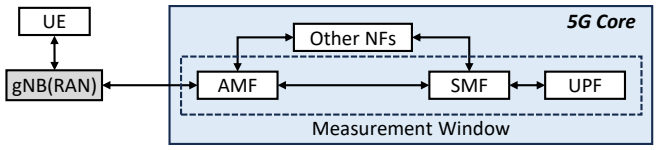
3GPP TS 28.552 defines standardized E2E Key Performance Indicators (KPIs), including Registration Time, PDU Session Establishment Time, Throughput, and Success Rate. These KPIs have been the dominant methodology in both standardization and research [6]. They are effective for capturing overall service quality but insufficient for diagnosing the internal sources of latency. Figure 1 illustrates how these KPIs cover only the start and end of procedures, leaving intermediate API-level delays hidden in a scalar measurement.

Beyond KPIs, prior studies have investigated low-level behaviors such as CPU and memory utilization, system calls, and NF-level queuing latencies [7]–[9]. While these efforts provide useful insight into the runtime characteristics of individual NFs, they do not capture how latency propagates across APIs in SBA workflows. As a result, existing approaches fall into two extremes: E2E KPIs aggregate too broadly and conceal where delays arise, whereas NF-level profiling focuses too narrowly and cannot explain inter-service interactions. What is missing is a methodology that exposes latency at the API level, where both the origins of performance bottlenecks and their impact on E2E service quality become visible.




*Registration Time = T(Registration Request → Registration Complete)*
(AMF) (AMF)

(a) Registration Procedure




*PDU Session Setup Time = T(Update SM Context → Create/Update SM Context)*
(SMF) (AMF)

(b) PDU Session Setup Procedure

Fig. 1: Mapping E2E KPIs to 5G Core procedures. (a) Registration Time to Registration procedure. (b) PDU Session Setup Time to Session Setup procedure.

### C. Challenges in Benchmarking the 5G Core Control Plane

Despite recent progress, benchmarking the 5G Core Control Plane still faces three critical and persistent challenges.

First, there is a lack of granularity in existing latency metrics. E2E KPIs collapse multi-step Control Plane procedures into aggregated single values. As a result, even if latency increases significantly and unpredictably, it is impossible to determine which NF or which API is truly responsible. This lack of fine-grained resolution ultimately prevents accurate identification of meaningful optimization targets.

Second, the experimental setup is too heavyweight, complex, and resource-intensive. Conventional evaluations require UE/RAN simulators and full Core deployments. Such setups tightly couple RAN and Core processing, making it extremely difficult to reproduce results, isolate subtle performance issues, or conduct rapid experimental iterations. They also consume excessive hardware and software resources, creating unnecessary operational overhead for Control Plane–only studies.

Third, existing benchmarking tools are functionally inadequate for realistic SBA. Generic HTTP load generators cannot accurately model 5G-specific workflows, since they lack support for dynamic NF discovery, multi-step API dependencies, and cryptographic operations such as 5G-AKA authentication. As a result, they only enable simplified mock tests that fail to faithfully reflect real Control Plane behavior.

In sum, current approaches either aggregate too much information, require excessive infrastructure components, or ignore 5G-specific API semantics. A lightweight, domain-specific methodology for precise API-level benchmarking is therefore essential for accurate, reproducible, and practically useful evaluation of 5G Core Control Plane performance.

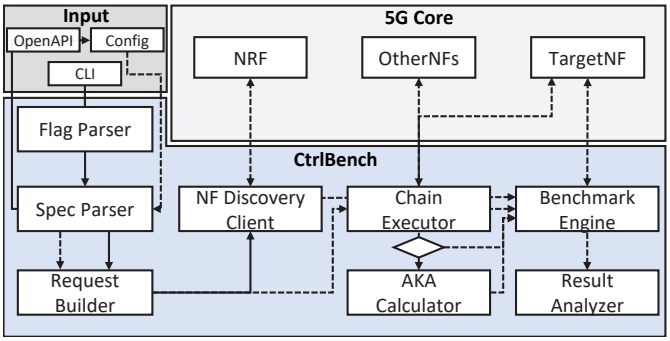



Fig. 2: Architecture of *CtrlBench*. Inputs are parsed into requests, NF endpoints discovered via NRF, APIs executed with AKA support, and results analyzed for latency attribution.

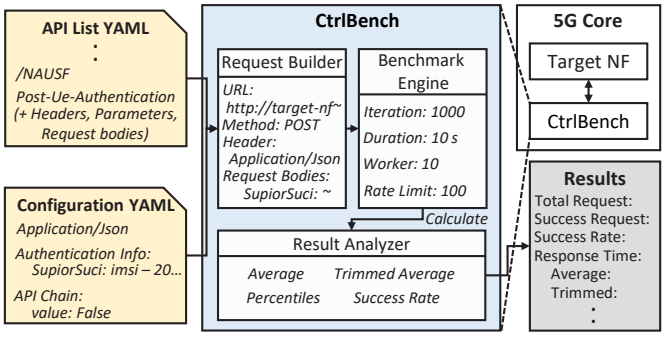



Fig. 3: API-level microbenchmarking in *CtrlBench*, from YAML input through request execution to result analysis.

## III. *CtrlBench* Design

This section presents the architecture and operational workflow of *CtrlBench*, which addresses key limitations in the performance evaluation of the 5G Core Control Plane.

### A. System Overview

**Overall Architecture.** *CtrlBench* is implemented as a lightweight, standalone, and domain-specific benchmarking tool that interacts with the 5G Core exclusively through standardized APIs. As illustrated in Fig. 2, the system consists of three logical layers: the input layer, the *CtrlBench* core, and the target layer. The input layer specifies detailed benchmarking configurations through CLI flags, OpenAPI specifications, and YAML templates that capture user-specified parameters. The core layer orchestrates benchmarking by managing request construction, NF discovery, prerequisite chaining, authentication, and result aggregation. The target layer represents the actual 5G Core NFs under test. This separation of concerns ensures that benchmarking remains portable, reproducible, and independent of particular NF implementations.

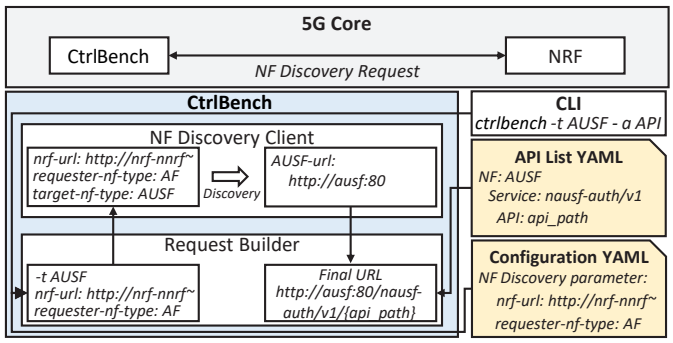
**Overall Workflow.** The workflow is divided into two phases: configuration generation and benchmark execution. In the configuration phase, *CtrlBench* parses OpenAPI specifications to extract detailed API metadata and generates YAML templates with placeholders for parameters, headers, and request bodies. For AUSF benchmarking, the templates incorporate mandatory fields for 5G-AKA credentials, enabling fully reproducible authentication tests. During execution, the tool loads the configuration, validates inputs, resolves NF endpoints via the NRF, and constructs API requests with correctly substituted parameters. When chain execution is required, prerequisite APIs are automatically invoked and their outputs injected into subsequent requests. The benchmark engine generates controlled load according to user-specified concurrency and rate limits, while the result analyzer aggregates response times, percentiles, and success rates. For APIs requiring authentication, the embedded 5G-AKA module computes RES* values and supplies them automatically to complete the authentication workflow.

### B. API-Level Performance Isolation and Reproducibility

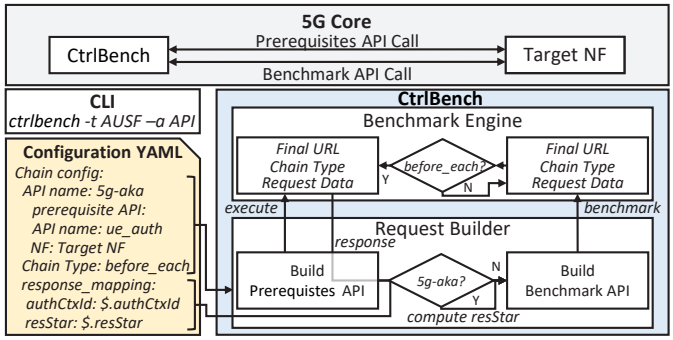
As shown in Fig. 3, *CtrlBench* isolates and precisely measures individual API calls, thereby overcoming the well-known limitations of conventional E2E metrics that collapse multi-step latencies into a single aggregate value and obscure the actual source of performance variation. Rather than simulating complete UE registration procedures, the tool directly invokes target APIs with predefined parameters, which effectively eliminates confounding factors such as RAN simulation overhead, unrelated inter-NF delays, or fluctuations introduced by complex signaling sequences. This strict isolation enables precise attribution of latency to specific NFs and, more granularly, to individual API calls. Response times are measured by capturing high-resolution timestamps immediately before request transmission and immediately after response reception.

Response-time variability is addressed through systematic statistical characterization instead of relying on single-point summary metrics. *CtrlBench* reports multiple percentiles (P50, P75, P90, P95, P99) and trimmed means (10%, 5%, 1%) to mitigate the disproportionate influence of outliers. In concurrent execution scenarios, results from all benchmarking workers are aggregated into unified latency distributions with corresponding statistical summaries. Successful and failed requests are recorded separately to prevent error responses from artificially distorting latency measurements.

Reproducibility is guaranteed by declaratively recording all parameters in version-controllable YAML files, including request bodies, headers, and chain relationships that capture the experimental state. The tool is distributed as a single compact binary with no runtime dependencies on databases, queues, or orchestration frameworks, ensuring reproduction of load patterns under the same configuration and CLI flags. Runtime parameters such as concurrency, duration, and throughput can be injected through CLI flags, enabling straightforward scripting of diverse scenarios. Results are exported in plain text for easy parsing and seamless integration into external analysis pipelines. Through API-level isolation and explicit configuration recording, *CtrlBench* transforms performance evaluation from a coarse-grained, environment-dependent process into a fine-grained, repeatable experiment, thereby addressing a fundamental gap left by conventional E2E simulators.




(a) NRF-based NF discovery automatically resolves NF endpoints for requests.




(b) API chaining executes prerequisite calls and computes authentication values, ensuring execution and realistic benchmarking of 5G Core procedures.

Fig. 4: 5G domain-specific features in *CtrlBench*.

### C. 5G Domain-Specific Feature Integration

Unlike general-purpose HTTP benchmarking tools that rely exclusively on static endpoints, *CtrlBench* incorporates 5G-specific features defined by 3GPP in order to capture the inherently dynamic characteristics of SBA deployments. As illustrated in Fig. 4(a), it implements NRF-based service discovery (TS 29.510), thereby allowing benchmark requests to adapt to runtime NF registration and evolving topology changes. The NF Discovery Client constructs queries with *target-nf-type* and *requester-nf-type*, parses *SearchResult* responses to extract NF profiles, and derives endpoints from the `ipEndPoints` array. Results are cached within a single benchmark execution to reduce unnecessary overhead and are invalidated upon completion to maintain fidelity and accuracy across independent runs. This design ensures that performance evaluation faithfully reflects operational 5G dynamics rather than relying on static or unrealistic assumptions.

Service workflows in 5G frequently involve sequential dependencies, where the outputs of one API invocation become inputs to another [10]. As depicted in Fig. 4(b), *CtrlBench* addresses this requirement through a chain execution mechanism declaratively defined in YAML. Each chain specifies a prerequisite API together with mapping rules, which employ JSONPath to extract response fields and substitute them into parameters, headers, or request bodies of subsequent invocations. Two execution modes provide methodological flexibility: `once_before_benchmark`, which executes the prerequisite once and reuses results throughout all iterations, and `before_each_call`, which ensures fresh values for every invocation to emulate independent transactions. Beyond simple extraction, computed values such as RES* can be stored in the `ExtractedData` map, thereby enabling cryptographic or stateful outputs to be seamlessly integrated into subsequent requests. This mechanism captures realistic service workflows while simultaneously preserving experimental rigor and reproducibility guarantees.

Furthermore, *CtrlBench* integrates domain-specific authentication logic that cannot be accommodated by generic benchmarking tools. The AUSF procedure necessitates cryptographic operations specified in TS 33.501, including Milenage-based verification and key derivation. Upon receiving RAND and AUTN from the `Post-Ue-Authentications` API, the tool derives SQN using AK, verifies AUTN through MAC checking, computes RES via function f2, and subsequently derives RES* for confirmation. This sequence is transparently executed during chain execution, with the resulting RES* automatically supplied to the `5G-Aka-Confirmation` API. By embedding authentication primitives within the benchmarking process, *CtrlBench* eliminates reliance on external scripts and ensures both automation and fidelity in evaluating security-critical procedures. This domain-specific integration, rooted in deliberate design choices, bridges the gap between generic benchmarking utilities and operational 5G Core workflows, establishing *CtrlBench* not merely as a synthetic traffic generator but as a domain-aware tool aligned with 3GPP specifications. By combining dynamic discovery, API chaining, and embedded authentication, it faithfully reproduces operational SBA workflows while ensuring academically reproducible results.

## IV. Implementation

*CtrlBench* is implemented in Go with custom support for 5G Control Plane benchmarking. Since many testbeds use HTTP/2 without TLS (h2c) [2], [11], we modified Go's `http2.Transport` to allow h2c by enabling `AllowHTTP` and replacing `DialTLS` with a TCP dialer. This enables practical interoperability while preserving optional TLS.

To emulate SBA workflows, *CtrlBench* supports declarative API chaining. Users specify prerequisite APIs and JSONPath extraction rules in YAML, and extracted values are automatically injected into target requests. Two execution modes are provided: cached once before benchmarking or re-execution before each call. For authentication, *CtrlBench* embeds 5G-AKA computations using the Free5GC Milenage library [12], performing SQN recovery, AUTN verification, and RES* derivation as defined in 3GPP TS 33.501 [13].

The load generator employs a worker pool with independent clients to avoid contention and track latency per worker. A token bucket algorithm enforces throughput limits, and results are aggregated into percentiles, trimmed averages, and success/failure counts. This ensures reproducible and fine-grained performance measurements at the API level.

## V. Evaluation

This section evaluates the effectiveness of *CtrlBench* in identifying performance variation points at the NF and API levels that remain concealed when using conventional E2E metrics. The objective is to determine whether *CtrlBench* can more clearly reveal the specific locations of latency introduced by deployment changes or performance anomalies.

### A. Evaluation Setup

To evaluate its effectiveness, three scenarios were constructed. **The baseline scenario (S1)** deployed Control Plane NFs across two worker nodes on a single physical host, with AMF and SMF co-located on one node and AUSF, PCF, NSSF, NRF, UDM, and UDR on the other. **The split-NF scenario (S2)** migrated the latter group to a worker node on a separate host, introducing additional network latency to their APIs. **The injected-delay scenario (S3)** extended S1 by adding artificial delay to the AUSF `Ue-Authentications` API, emulating anomalies observed in practice.

Experiments were conducted on a Kubernetes cluster comprising five virtual machines across two physical hosts interconnected with 10 Gbps links. Nodes ran Ubuntu 22.04 with Cilium as the default CNI. Free5GC was used as the 5G Core and UERANSIM as the UE/RAN simulator [12], [14].

Two load conditions were considered: single UE registration and multiple UE registrations. In the latter, ten registration requests per second were issued until 1,000 UEs were registered. Measurements focused on three representative APIs: `UE-Authentications` and `5G-AKA-Confirmation` from the AUSF, and `Generate-Auth-Data` from the UDM. To approximate realistic deployment, *CtrlBench* was co-located with the AMF for AUSF APIs and with the AUSF for the UDM API. API-level latency measurements were compared against the E2E Registration Time.

### B. Execution Workflow of Benchmarking

Fig. 5 illustrates the detailed execution workflow of benchmarking the AUSF's 5G-AKA-Confirmation API using *CtrlBench*. The process begins by parsing the OpenAPI specification to generate benchmarking templates, after which the benchmark is initiated through the CLI. Based on NRF-discovered information, the tool automatically derives the API endpoint and request parameters, ensuring that the constructed HTTP request reflects the actual service topology. A complete request is then generated, including body fields and protocol headers. Prior to invocation, the RES* value mandated by the 5G-AKA procedure is computed through embedded cryptographic functions, thereby enabling authentication-compliant execution without external dependencies. Finally, the API call is issued and both the response semantics and latency are reported as benchmarking results.

This workflow highlights that *CtrlBench* goes beyond conventional load generators by embedding 3GPP-standardized procedures directly into the benchmarking process. By integrating service discovery, parameter substitution, and authentication primitives, the tool provides operationally faithful measurements that align with real 5G Core behavior, rather than relying on static or artificially simplified request templates.

```
> ctrlbench -b ausf
Building configuration file...
Building configuration for specified NFs: [AUSF]
Valid NFs found: [AUSF]
Configuration file created: configuration.yaml
API list file created: openapi/api_list.yaml
```

(a) Parsing the OpenAPI specification and generating templates

```
> ctrlbench -t ausf -a PutUeAuthentications5gAkaConfirmation -d 10 -r 3
Starting PrepareAPIExecution for NF=AUSF, API=PutUeAuthentications5gAka
Confirmation, skipChain=true
Skipping chain check for prerequisite API: PutUeAuthentications5gAkaCon
firmation
Using placeholder for field 'resStar' that will be filled by before_eac
h_call chain
Configuration validation passed - ready for execution
```

(b) Executing a sequential benchmark through the CLI

```
Discovered AUSF URL: http://controlplane-free5gc-ausf-service:80
Replaced path parameter: {authCtxId} -> imsi-208930000000003
Starting sequential benchmark:
   Duration: 10s
   Rate Limit: 3 req/s
Header[Content-Type] = application/json
Header[Accept] = application/json
Replaced path parameter: {authCtxId} -> imsi-208930000000003
Final URL: http://controlplane-free5gc-ausf-service:80/nausf-auth/v1/ue
-authentications/imsi-208930000000003/5g-aka-confirmation
Execution Details:
   NF: AUSF
   API: PutUeAuthentications5gAkaConfirmation
   Method: PUT
   Path: /nausf-auth/v1/ue-authentications/{authCtxId}/5g-aka-confirmat
ion
   Discovered URL: http://controlplane-free5gc-ausf-service:80
   Parameters: map[authCtxId:imsi-208930000000003]
Request Body: {"resStar":"PLACEHOLDER_FOR_CHAIN_EXECUTION"}
```

(c) Building the complete HTTP request

```
========== FREE5GC RESSTAR CALCULATION ==========
========== CALCULATION COMPLETE ==========
Computed XRES* (resStar): 8f28022844bd665b8006ff306aeef6ee
Computed resStar: 8f28022844bd665b8006ff306aeef6ee
Prerequisite API completed for before_each_call
ExtractedData available, contents:
resStar: 8f28022844bd665b8006ff306aeef6ee
Found value using key 'resStar': 8f28022844bd665b8006ff306aeef6ee
Applied request body mapping: resStar = 8f28022844bd665b8006ff306aeef6e
e (from $.resStar)
Configuration validation passed - ready for execution
Replaced path parameter: {authCtxId} -> imsi-208930000000003
Response Body: {"authResult":"AUTHENTICATION_SUCCESS","supi":"imsi-2089
30000000003","kseaf":"8f7bd47fcce6ad3f98f529247051b966332b6fe509bbb43d1
1d040442af44df8"}
```

(d) Computing RES* for 5G-AKA confirmation

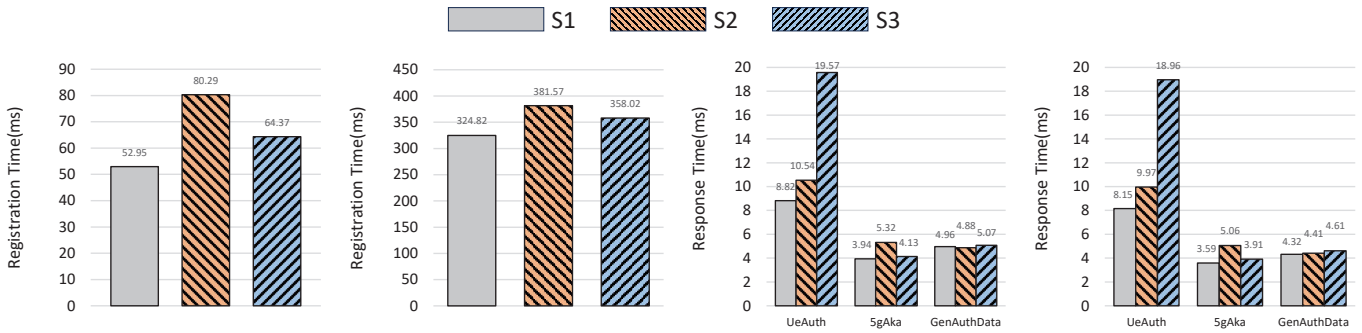
```
============================================================
  SEQUENTIAL BENCHMARK RESULTS
============================================================
Total Requests: 29
Successful: 29
Failed: 0
Success Rate: 100.00%
Total Duration: 10.001000728s

Response Times:
Average: 5.651477ms
Minimum: 5.013926ms
Maximum: 6.749976ms
90% Trimmed Average: 5.634405ms
95% Trimmed Average: 5.634405ms
99% Trimmed Average: 5.651477ms
```

(e) Reporting the benchmark result of the API call

Fig. 5: Execution workflow of *CtrlBench* benchmarking the AUSF's 5G-AKA Confirmation API.

### C. Analysis of E2E and API-Level Metrics

In the single-UE registration experiment (Fig.6(a, c)), the average Registration Time was measured as 52.95 ms, 80.29 ms, and 64.37 ms for scenarios S1, S2, and S3, respectively. During the same runs, the UE-Authentication, 5G-AKA-Confirmation, and Generate-Auth-Data APIs exhibited latencies in the narrow range of 3.94–19.57 ms, remaining stable across scenarios. When the load was scaled to 1,000 UEs




(a) Single-UE Registration Time (b) Registration Time under 1000 UEs (c) Single-call API latency (d) API latency under 1000 calls

Fig. 6: Evaluation Results of E2E Registration Time and API Response Time in Three Scenarios

(Fig.6(b, d)), the Registration Time rose sharply to 324.82 ms, 381.57 ms, and 358.02 ms, representing more than a fivefold increase. In contrast, the latencies of the three representative APIs stayed within 3.59–18.96 ms, without reflecting the same level of growth observed at the E2E level.

These results indicate that E2E metrics capture a substantial degradation of the overall registration procedure under heavy load, whereas API-level measurements demonstrate that the latency of individual service interfaces remains comparatively stable. The discrepancy shows that the increase in Registration Time is not explained by the simple accumulation of API processing delays, but instead arises from procedural complexity and systemic bottlenecks across the workflow.

Moreover, scenario-specific differences further illustrate the analytical capability of *CtrlBench*. In S2, where NFs were distributed across physical hosts, the UE-Authentication and 5G-AKA-Confirmation APIs experienced moderate latency increases relative to S1, reflecting the cost of cross-host communication. In S3, the artificial delay injected into the AUSF is directly captured as elevated latency for the UE-Authentication API, while the unrelated APIs remain unaffected. This selective sensitivity highlights that *CtrlBench* not only identifies performance degradation at the level of the complete procedure but also attributes delays to specific NF placements and API paths.

## VI. Conclusion

This paper introduced *CtrlBench*, an OpenAPI-driven microbenchmarking framework for the 5G Core Control Plane that addresses the limitations of E2E metrics by exposing latency at individual service interfaces and isolating API-specific bottlenecks. Evaluations across single-UE, large-scale registration, and NF deployment scenarios showed that *CtrlBench* uncovers performance variations invisible to aggregated metrics, demonstrating both accuracy and scalability for practical 5G Core analysis. By enabling lightweight, reproducible, and domain-specific benchmarking without requiring full UE/RAN simulation, it establishes a foundation for systematic diagnosis and optimization of service-based architectures. Future work will extend *CtrlBench* to capture richer workflows with conditional dependencies, parallel invocations, and complex service compositions, thereby providing deeper insights for both current 5G deployments and emerging 6G networks. Furthermore beyond benchmarking, *CtrlBench* can also support regression testing and anomaly localization pipelines, enabling API-aware performance engineering and fair cross-deployment comparisons.

## Acknowledgment

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT). (No.RS-2024-00398379, Development of High Available and High Performance 6G Cross Cloud Infrastructure Technology)

## References

[1] "TS 23.501," https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf.
[2] "TS 29.500," https://www.etsi.org/deliver/etsi_ts/129500_129599/129500/16.04.00_60/ts_129500v160400p.pdf.
[3] ITU-R, "Recommendation ITU-R M.2083-0: IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond," https://www.itu.int/rec/R-REC-M.2083, September 2015.
[4] "TS 29.510," https://www.etsi.org/deliver/etsi_ts/129500_129599/129510/17.06.00_60/ts_129510v170600p.pdf.
[5] "TS 28.552," https://www.etsi.org/deliver/etsi_ts/128500_128599/128552/16.06.00_60/ts_128552v160600p.pdf.
[6] Y.-S. Liu, S. Qi, P.-Y. Lin, H.-S. Tsai, K. K. Ramakrishnan, and J.-C. Chen, "L25gc+: An improved, 3gpp-compliant 5g core for low-latency control plane operations," in *2023 IEEE 12th International Conference on Cloud Networking*, 2023, pp. 203–211.
[7] M. Barbosa, M. Silva, E. Cavalcanti, and K. Dias, "Open-source 5g core platforms: A low-cost solution and performance evaluation," in *2025 International Conference on Information Networking*, 2025, pp. 99–104.
[8] T. Mukute, L. Mamushiane, A. A. Lysko, E.-R. Modroiu, T. Magedanz, and J. Mwangama, "Control plane performance benchmarking and feature analysis of popular open-source 5g core networks: Openairinterface, open5gs, and free5gc," *IEEE Access*, vol. 12, pp. 113 336–113 360, 2024.
[9] S. Subramanian, M. R. Kanagarathinam, and K. M. Sivalingam, "Performance evaluation of 5g core network control-plane using open5gs and kubernetes," in *2025 17th International Conference on COMmunication Systems and NETworks*, 2025, pp. 584–592.
[10] "TS 23.502," https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/16.07.00_60/ts_123502v160700p.pdf.
[11] "RFC7540," https://datatracker.ietf.org/doc/html/rfc7540.
[12] "free5GC," https://free5gc.org/.
[13] "TS 33.501," https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/16.03.00_60/ts_133501v160300p.pdf.
[14] "UERANSIM," https://github.com/aligungr/UERANSIM.