Cloud5GC : Design and implementation of scalable and stateless mobile core system on public cloud

Kunio Akashi, Seiichi Yamamoto,
Haruki Sakurai, Koki Ito, Tomohiro Ishihara,
Takuji Iimura, Yuji Sekiya
The University of Tokyo
Tokyo, Japan
k-akashi@si.u-tokyo.ac.jp, yama@iis.u-tokyo.ac.jp,
harusaku0412@g.ecc.u-tokyo.ac.jp,
utmt0328@g.ecc.u-tokyo.ac.jp, sho@c.u-tokyo.ac.jp,
iimura-takuji@g.ecc.u-tokyo.ac.jp, sekiya@nc.u-tokyo.ac.jp

Hiroki Watanabe, Keiichi Shima, Katsuhiro Horiba,
Research Institute of Advanced Technology
SoftBank Corp.
Tokyo, Japan
hiroki.watanabe14@g.softbank.co.jp,
keiichi.shima@g.softbank.co.jp,
katsuhiro.horiba@g.softbank.co.jp

***Abstract*—For contemporary IT services, Wireless communication, facilitated by mobile phones, is a fundamental infrastructure. 5G comprises RAN (Radio Access Network) and Mobile Core (5GC) technologies. Specifically, 5GC handles communication from smartphones and IoT devices, enabling communication tailored to each use case, thereby solidifying its position as the Mobile Core system within the 5G network. With the recent proliferation of smartphones and IoT devices, along with advancements such as digital twins, there has been a substantial increase in traffic demand on 5GC. This surge has compelled telecom companies to invest heavily in equipment upgrades and operational maintenance to ensure stable 5GC communication. To address the challenges in the Mobile Core, our research has aimed at reimagining the 5GC architecture. We introduced Cloud5GC, a groundbreaking solution that transitions away from traditional function-based systematization towards a more cohesive micro Mobile core architecture. Implementing Cloud5GC on a public cloud platform demonstrated its flexibility, resilience, and fault-tolerance. Our findings underscore the architectural excellence of Cloud5GC, representing a significant advancement in 5G communications.**

I. Introduction

Mobile phones are essential in the modern IT world, with smartphones enabling not just calls but also data communication. The rise of IoT devices adds to the growing number of devices connecting to mobile networks each year. As mobile network technologies progress, the shift from 4G/LTE to 5G is underway. The 5G system is divided into two systems; the Radio Access Network (RAN) and the Core Network (CN). The RAN system includes mobile devices and base stations, mainly for radio wave transmission. CN is composed of the control network, called “Control plane” (C-Plane), which handles User Equipment (UE) registration, billing, and state management, and “User plane” (U-Plane), which manages data communication from UEs to the Internet. In other words, the Core system is a critical part of a mobile network because a single Core manages tens of millions of UEs. On the other hand, the gNodeB (gNB) systems are installed based on the reach of their radio waves, inevitably resulting in a fewer number of UEs managed by a single gNB.

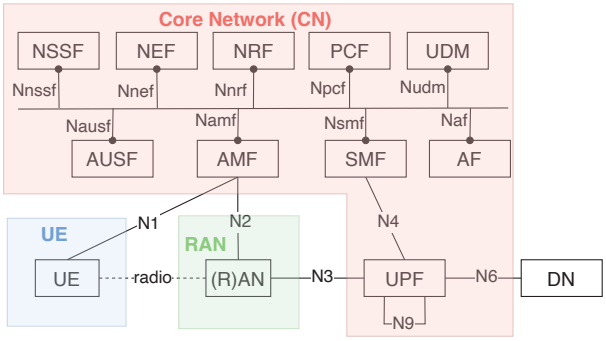

Fig. 1: Overview of a typical 5G system

Both the RAN and the CN are essential components of the 5G system. However, a failure in the CN typically leads to a severe and potentially catastrophic outage in the 5G system. This is because, as previously mentioned, the RAN consists of numerous gNBs. If one of these gNBs experiences a failure, only the UEs managed by that specific gNB are affected by the issue. Moreover, even if a failure occurs in a single gNB, as long as the signal from neighboring gNBs reaches the UE, the UE will switch to a functioning gNB to maintain communication, preventing a complete communication breakdown.

On the other hand, if a failure occurs in the CN, all the gNBs connected to that CN are impacted, and consequently, all UEs linked to those gNBs lose communication capabilities. Such a failure in the CN can be catastrophic for the 5G system. Mobile communication companies invest heavily in system redundancy to prevent CN failures. This stems from the fact that the mobile network was built following the legacy of the circuit-switched model used in traditional telephony. The CN acts as a system that aggregates and centrally manages endpoints, or UEs, a model that has remained consistent from the era of the second-generation GSM, through 3G, 4G/LTE, and into the current 5G. As the number of devices connected to the mobile network and the volume of communication continue to increase annually, the cost to maintain the CN

system has also increased. The potential impact scope of a failure in the CN system has expanded, and real incidents have occurred in Japan [1], leading to significant societal concerns.

In this study, rather than relying on the traditional centralized and large-scale Core system, we designed and implemented a micro Mobile Core architecture named Cloud5GC. The proposed micro Mobile Core architecture executes serverless functions on a per-procedure code basis, rather than distributed execution on a per-function basis as designed in traditional 5GC. This ensures that only a single UE is affected during a CN process failure. In addition, the micro Mobile Core architecture is designed as deployment on AWS Lambda, making it easy to scale out and ensuring redundancy. This approach aims to significantly enhance the scalability, redundancy of the Mobile Core. We further evaluated its performance and benefits.

The structure of this paper is as follows: Section II discusses related works; Section III elaborates on the design philosophy of Cloud5GC. Further, Section IV and V provide details on the design and implementation of Cloud5GC, while Section VI presents the evaluation results. Finally, Section VII concludes this study.

II. Related works

This section describes related studies on Mobile Core systems and summarizes the differences from this study.

"Investigating Inter-NF Dependencies in Cloud-Native 5G Core Networks" [2] describes the Service-Based Architecture (SBA) in CNs. In this paper, SBA achieves flexibility and agility in a move towards cloud-native implementations. However, the advanced functional decomposition in SBA implies an increase in Network Function (NF) signaling traffic during the execution of C-Plane procedures, amplified overheads from serialization, and complexities in ensuring UE state consistency and orchestration in systems where NF dependencies are tight. Furthermore, the Stream Control Transmission Protocol (SCTP), unchanged from the 4G era, is utilized for the connection between RAN and CNs, presenting challenges due to its low compatibility with load balancers in public clouds, scalability concerning the number of connectable RAN devices, and significant impacts from temporary transport network disconnections.

PP5GS [3] discusses the challenges and implications of adopting SBA for CN. PP5GS addresses the challenges of edge-offloading in 5G Core Networks by introducing a procedure-based and stateless architecture. This design enables the deployment of self-contained Per-Procedure Network Functions (PPNFs) at the edge, subsequently reducing inter-NF communication and signaling overhead. Unlike the traditional SBA, PP5GS minimizes the need for communication between NFs by integrating the processing logic of a C-Plane procedure into a single PPNF.

ECHO [4] adopts an SCTP proxy to leverage the load balancer functionality of public clouds, aiming to achieve a scalable CN.

MAGMA [5], on the other hand, defines a consolidated NF called AGW, which integrates AMF, SMF, and UPF. This not only functions as an SCTP proxy but also reduces C-Plane traffic between NFs. Additionally, within the AGW, modules for Mobility Management and Session Management, as well as interactions with other NFs, utilize gRPC. This approach minimizes signaling overhead and ensures resilience against transient transport network disconnections.

Organic 6G Core [6] focuses on the complexity of inter-NF coordination brought about by the SBA in 5GC and presents a design for the 6G core network that revisits the concept of functional split.

Procedure-based 5GC (Proc5GC) [7] shares with PP5GS and Cloud5GC the approach of re-focusing on the steps to redesign the functional partitioning of the Mobile Core. Furthermore, like Organic 6G Core, it delves into the complexity of the Mobile Core and discusses the prerequisites for solving it. Proc5GC not only meets these prerequisites, but also addresses the SCTP challenge by introducing SCTP proxies, similar to the ECHO and MAGMA approaches. However, Proc5GC differentiates itself by placing a message queue between the SCTP proxy and the various functions of the Mobile Core. This setup allows N1/N2 message transmission using the Pub/Sub method and enables event-driven processing on a message-by-message basis.

III. Our goals

Summarizing the issues with the Mobile Core as discussed in Sections I and II, we identify the following three problems.

- P-1 Current designs are not ideal for data communications because they inherit concepts from the era of telephone circuit-switching networks. As a result, it lacks scalability and redundancy.
- P-2 To accommodate a large number of UEs, the Mobile Core system has been scaled up significantly. This means that in the event of a failure, the impact range is considerable, and the construction and operation entail substantial costs.
- P-3 The functions within the Mobile Core are fragmented. To manage the state of UEs, there's frequent state message exchanges between functions, leading to significant messaging overhead.

To realize these design objectives, our study did not simply implement the functionalities required by the Mobile Core based on the definitions provided by 3GPP and then combine them. Instead, we designed a procedural Mobile Core where functionalities are activated on-demand according to the requests from UEs. Essentially, for each UE, it works as if an exclusive Mobile Core is instantiated, allowing processes to be isolated per UE. The idea and preliminary measurement results were published in Proc5GC. The paper discusses the limitations of the current CN architecture and proposes a novel architecture, Proc5GC to address these issues. Proc5GC focuses on providing stateless functional processing for each message of a 5GS procedure, eliminating the need for context maintenance across multiple NFs.

Proc5GC introduces several key features that contribute to its improved performance and scalability compared to the traditional NF-based 5GC architecture.

- Separation of SCTP endpoints from CN: Proc5GC installs a module called n1n2gw next to gNB, separating the SCTP endpoint from the CN. This allows for asynchronous communication between the RAN and CN, reducing the spread of faults and improving fault tolerance.
- Single program for each CN functionality: In Proc5GC, each CN functionality is provided as an independent functional program for each message. This stateless design allows easier implementation of new functionalities and reduces the side effects of adding new features.
- Instantiation of CN functionality for each message arrival: Proc5GC instantiates and executes a stateless functional program for each message arrival. This ensures that the impact of software-specific problems or processing issues is localized to a single UE or RAN, improving performance and scalability.
- Sharing a single GUAMI within a CN: Proc5GC allows for a single Global Unique AMF Identifier (GUAMI) to be shared within a CN, regardless of the n1n2gw or Message Queue (MQ) instance. This eliminates the need for UEs to switch AMFs, reducing signaling and context migration during inter-AMF handover and maintenance.

These key features of Proc5GC address the limitations of the NF-based 5GC architecture and contribute to improved performance, fault tolerance, and scalability.

To improve the scalability and redundancy of the Proc5GC design, in this study we tried to implement each procedure in Proc5GC as a micro Mobile Cores. The design requirements for micro Mobile Core are outlined as R-1 to R-3:

R-1 Ensuring scalability across the entire Mobile Core, it is designed such that the required number of micro Mobile Cores can be initiated and executed in parallel, based on the number of UE connections.

R-2 Enhancing the overall redundancy of the Mobile Core, the implementation is partitioned by processing units, ensuring that one process implementation does not influence another.

R-3 Minimizing the resources used by the entire Mobile Core, it is designed to only activate the functions necessary to address a request from the UE, thereby eliminating resource wastage.

By fulfilling these requirements, it became possible to deploy Proc5GC on a per-UE basis. This allowed state management specific to each UE, eliminating the need for state sharing between UEs and realizing a truly stateless Mobile Core. As a result, the Mobile Core could be built in the Public Cloud using a Serverless model.

IV. Design

In this study, to fulfill the design requirements R-1 to R-3 discussed in the previous section, we designed and implemented Cloud5GC, which dynamically launches a micro Mobile Core containing only the functions necessary for processing requests from a UE.

To evaluate the feasibility of Cloud5GC, we designed and implemented C-Plane of the Core system mentioned in Section I. This section describes the detailed design and implementation. Figure 2 shows the proposed architecture of C-Plane in Cloud5GC.

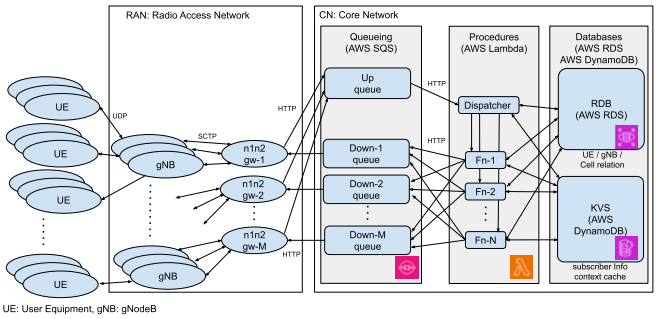

Fig. 2: The structure of Cloud5GC

As shown in Figure 2, Cloud5GC is implemented without using centralized control servers, employing a serverless architecture to implement various functionalities. Moreover, for inter-functional data exchanges invoked within the serverless architecture, we utilized a message queue and Key Value Store (KVS), common in web systems. The "Queuing" module establishes connections between RAN and Core, and the "Procedures" module activates and connects only the necessary functions in response to the UE's processing requests.

Figure 3 shows the concept of micro Mobile Core architecture in Cloud5GC. A dedicated Mobile Core is initiated for each UE, and upon completion of the process, it terminates. Data required for processing is stored in databases. Only the necessary information is read and written by each function, It enables stateless processing without continually retaining information in processes. The details of how we met the requirements are described below.

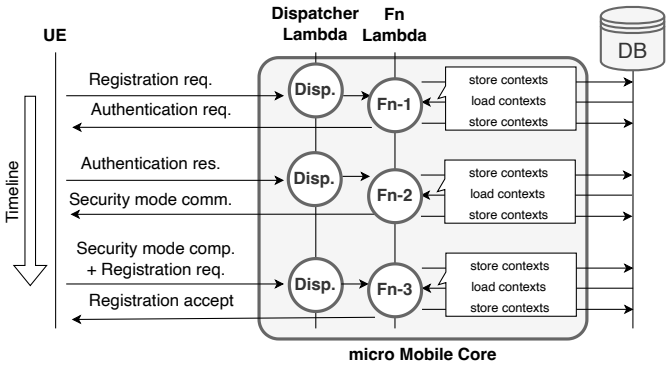

Fig. 3: The Concept of micro Mobile Core

A. Design for Scalability

We explored methods for executing Mobile Core processes on a public cloud with excellent scalability and parallel

execution capabilities to fulfill the requirement R-1. Instead of implementing them on VMs within IaaS to maximize cloud benefits, we chose to deploy them on PaaS. When using IaaS and implementing the Mobile Core on a VM, one would either need to enhance a single VM's resources or initiate multiple VMs for load distribution, similar to traditional Mobile Core scaling methods. This approach involves high costs and complexity for VM startup, termination, and management, making rapid scalability challenging. Consequently, our research leveraged serverless execution environments, a feature of public cloud PaaS, as the foundation for the Mobile Core. This allowed us to instantly initiate only the necessary functions in response to UE requests, facilitating fast start-ups, terminations, and numerous parallel executions. Specifically, we implemented using AWS Lambda [8] functions. Only required functions responding to UE requests were implemented as multiple Lambda functions, which were then integrated using message queues such as Amazon SQS [9], realizing Mobile Core functions using only AWS PaaS capabilities.

B. Design for Redundancy

We designed a dedicated micro Mobile Core on a per-UE basis, which activates only the necessary functions for the operation and manages the state solely on a per-UE basis, we employed two types of databases. These are the KVS, a high-speed database, and the RDB, a database that manages information based on a schema.

The KVS was deployed to parameters related to the UE's status. As a result, there is no longer a need for the Mobile Core itself to continuously hold information or to transfer information from one function to another using APIs or message queues. When the Mobile Core executes a procedure, it activates only the necessary functions temporally, retrieves the required information from the KVS, and stores the parameters which may be used in the next procedure, then deletes the procedure.

This approach realized a micro Mobile Core that processes on a per-request basis from the UE. On the other hand, the RDB was employed to manage static information, such as subscriber details and gNB data, which doesn't change frequently with UE processing. We deploy these two types of databases following the inherent characteristics of each: the KVS for temporary value reading and writing, and the RDB for reading fixed values. This approach allowed for the realization of a stateless micro Mobile Core that doesn't manage the UE's state internally.

V. Implementation

As shown in Figure 2, we implemented Cloud5GC as three modules; Queuing module, Procedures module, and Database module.

A. Procedure module

In Cloud5GC, same as Proc5GC, CN functions were implemented monolithically for each procedure. We refer to each implemented procedure as "Fn procedure". Additionally, we prepared a procedure exclusively for task dispatching to call each Fn procedure, which we named "dispatcher procedure". The dispatcher procedure checks the presence of the Initial Message in the NGAP-PDU and references the NGAP Type within the Initial Message to invoke the corresponding Fn procedure.

Since the procedures utilize a PaaS environment, cloud service providers employ horizontal scaling through their load balancing functionalities. Specifically, when the load on a particular procedure increases, the procedure process is automatically replicated, performing a scale-out operation to enhance the processing speed over a certain duration. Conversely, when the load on a procedure decreases, procedures in an idle state waiting for processing undergo a scale-in operation, where they are terminated after a specific idle time duration.

Each Fn procedure was monolithically implemented to achieve the following two features:

(1) Reducing Inter-process Messaging Overhead: The input for the procedure consists of N1 and N2 messages received through RAN and the state or context of UE and RAN stored in the DB. Since the procedure is monolithically implemented, it processes without any additional messaging among procedures.

(2) Reducing Operational Cost: There are no dependencies between procedures, making the Fn procedures in this system loosely coupled. An Fn procedure is invoked for each UE, so this allows for rolling updates to be performed on specific Fn procedures during system operations without service interruption.

B. Queueing module

The procedures in the Procedure module are processed swiftly; however, under heavy loads, the processing may experience delays. If the influx of messages to a procedure exceeds its processing capacity, some messages might be missed. To address this, a queue is positioned at the beginning of the procedure process, acting as a buffer to temporarily store incoming messages. This design concept is the same as Proc5GC.

Cloud5GC uses PaaS for procedure processing. When there's a spike in load, the procedure process is duplicated and created as a serverless function at the request of the UE. This scale-out process of duplication and creation can cause delays. If the influx of messages into a procedure exceeds the scaling speed of that procedure, some messages may not be processed. To avoid this, Cloud5GC introduces queues.

Also, we strategically positioned the queues to optimize processing performance. By using separate queues for requests from the UE to the Procedure module and for responses from the Procedure module to the UE, we enabled responses to the UE immediately upon completion without sequential processing. Furthermore, in the response process from the Procedure module to the UE, by detecting NGAP-PDU value within the response messages, it allows for targeted messages to route to the appropriate n1n2gw's queue. This approach enables the decentralization of response message processing.

C. Database module

We implemented a database module to facilitate processing for each UE and maintain the management state of each UE. By storing information in the database, the procedure process is allowed to terminate once their required tasks are completed. We utilized two types of databases: a Key Value Store (KVS) for temporary information and a Relational DataBase (RDB) for persistent data. This concept is the same as Proc5GC. In Cloud5GC, we use AWS DynamoDB [10] for KVS and AWS RDS [11] for RDB to implement the concept in a public cloud environment.

VI. Evaluation

We evaluated the performance of the Registration Procedure defined in 3GPP TS 23.502 [12] on Cloud5GC. Cloud5GC experiences significant latency because it communicates between the gNB, located at our university, and Cloud5GC on AWS via the Internet. Consequently, the performance of a single UE tends to be inferior compared to other 5GCs. However, Cloud5GC is designed to scale with the granularity of Procedure Functions. As the number of UEs increases, the decline in Cloud5GC's performance becomes less significant.

We conducted two experiments to evaluate the performance of Cloud5GC. In the first experiment, we measured the time taken from the Registration Request to the Registration Accept for a single UE and compared it to free5GC. Free5GC is an open-source 5GC developed by NCTU that is compatible with 3GPP Release15. This experiment is focused on evaluating whether the process is completed within the specified timeout period for the registration procedure, as defined by the 3GPP standard. In the second experiment, to assess scalability, we measured the time of the Registration Procedure for multiple UEs and compared it to free5GC.

For the experiments, we prepared three servers. The first one, running UERAMSIM [13], is a DELL PowerEdge R640 with Ubuntu Linux 22.04 LTS, two Intel Xeon Gold 6230 2.1GHz 20-Core CPUs, and 384GB memory. The second server, running the n1n2gw and free5GC, is a DELL PowerEdge R650 with Ubuntu Linux 22.04 LTS, two Intel Xeon Gold 6330N 2.2GHz 28-Core CPUs, and 512GB memory. The third server, running Spirent Landslide [14] for emulating UEs and gNB systems, is DELL PowerEdge R7524 with VMware ESXi 7.0.3, two AMD EPYC 7443 24-Core CPUs, and 512GB memory.

A. Processing Time for Registration Procedure

In the first experiment, we use UERANSIM to emulate a UE and a gNB system, focusing on evaluating the time for Registration Procedure. Table I presents the result of this experiment. The table illustrates the time duration for a single UE and gNB system to complete the Registration Procedure, which is a series of functions such as Authentication Request, Security Mode Command, and Registration Accept. While free5GC completes each step in approximately 20 ms or less, Cloud5GC requires about 250 ms to 350 ms. As a result, Cloud5GC requires a longer duration to complete the Registration Procedure compared to free5GC. However, since the Registration was completed within 4 seconds, which corresponds to the 3GPP Registration Procedure timer, it indicates that Cloud5GC can provide service to the UE with sufficient speed.

Additionally, Figure 4 depicts the breakdown of the time required for processing in each serverless function. It becomes clear that in all serverless functions, activities like Context Restore/Store and RDB Get/Update, which involve database access, dominate the processing time. In other words, the Registration Procedure of Cloud5GC is heavily influenced by the time to access the DBs.

TABLE I: Registration Response Time (Single UE)

	Cloud5GC	free5GC
Authentication Request	250ms	26ms
Security Mode Command	591ms	36ms
Registration Accept	958ms	47ms

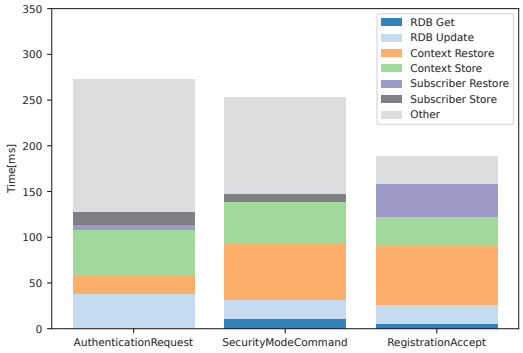

Fig. 4: Processing Time at Fn procedures (Single UE)

B. Scalability for Registration Procedure

In the second experiment, aimed at evaluating scalability, we measured and analyzed the processing times required to complete the Registration Procedure from the number of UEs and achieve the "Registration Complete" state. The results are illustrated in Figure 5. The x-axis indicates the time from the initiation of the "Initial UE message" to achieving the "Registration Complete" state, and the y-axis shows the CDF of completed registrations. In this experiment, we used Spirent LandSlide to emulate a large number of UEs and a gNB connecting them.

Cloud5GC scales the Procedure function based on requests from UEs, and it's evident that the more UEs present, the quicker Cloud5GC completes the Registration Procedure. This indicates that the micro Mobile Core architecture is designed to scale out with the increasing number of UEs, as it triggers a serverless function for each UE request. In contrast, free5GC consistently encountered retries and never completed the registration for 1024 UEs, while Cloud5GC successfully processed 1024 UEs without any retries. However, there were occasions where Cloud5GC's Registration for 1024 UEs was delayed in reaching Acceptance state.

Based on these findings, even with the Write Capacity Unit (WCU) in DynamoDB increased from 1000(default) to 4000, write throttling occurred, causing delays during the UE's context Store/Restore. [15] Since it depends on the configuration of AWS DynamoDB, it is expected that performance will improve if a larger WCU is configured.

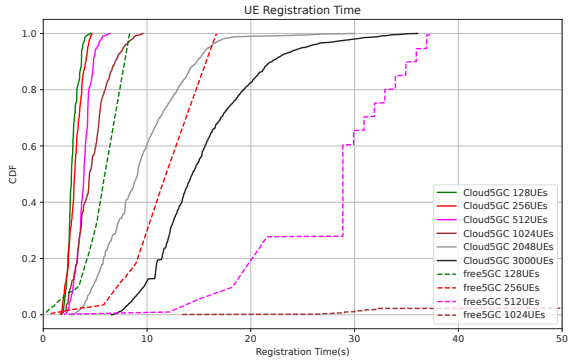

Fig. 5: Registration Completion Time (Cloud5GC vs free5GC)

VII. Discussion and Conclusion

PaaS is a serverless architecture, a model in which the cloud provider handles scaling and resource management. This model is based on the concept of deploying the necessary amount of resources when needed. From this, resource initialization work is required for the first time processing, and compared to processing after the first time, processing time is required only for the first time.

Public clouds have service specifications determined by the cloud providers, which implies certain limitations regarding resource utilization. Scalability is not infinite. While there are undeniable advantages in terms of scalability and fault tolerance when using a public cloud, it's clear that these benefits are not provided without bounds, as evidenced by our evaluation. Furthermore, public clouds can also experience outages. Therefore, when operating the Mobile Core on the cloud, relying solely on a single cloud provider can be risky.

Conventional 5G core architectures face the issues such as a large impact during some system failures and significant operational and maintenance costs. This is due to the Mobile Core system architecture supporting cellular networks inheriting the legacy telephone switching systems, aiming for a centralized management architecture. To address these issues, we proposed a new Mobile Core architecture called Proc5GC, which processes only the necessary tasks statelessly in response to requests from UEs. However, while Proc5GC is designed to run on on-premises servers or IaaS cloud environments, it has limitations in scalability and operability.

In this study, we introduce an improved Mobile Core architecture, called Cloud5GC. Compared to the conventional 5G core, Cloud5GC boasts superior scalability, fault tolerance, and resource efficiency. To validate the feasibility of the proposed architecture, we implemented the system on a public cloud environment and evaluated. Our results indicated that Cloud5GC could outperform compared to an existing 5G core implementation.

Acknowledgement

We would like to express our deepest gratitude to the "Spirent Landslide" product. Additionally, our sincere thanks go to Mr. Akihiro Nakamura and Mr. Masayuki Takemura from Spirent Communications for their technical support in emulating UEs and gNB systems on Landslide. We also thank Mr. Kentaro Someya from AWS Japan for his technical support to use AWS.

References

[1] KDDI. The July 2 Communication Failure and Our Response, 2022. https://www.kddi.com/english/important-news/20220729_01/.

[2] Endri Goshi, Michael Jarschel, Rastin Pries, Mu He, and Wolfgang Kellerer. Investigating inter-nf dependencies in cloud-native 5g core networks. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 370–374, 2021.

[3] Endri Goshi, Raffael Stahl, Hasanin Harkous, Mu He, Rastin Pries, and Wolfgang Kellerer. Pp5gs – an efficient procedure-based and stateless architecture for next generation core networks. *IEEE Transactions on Network and Service Management*, pages 1–1, 2022.

[4] Binh Nguyen, Tian Zhang, Bozidar Radunovic, Ryan Stutsman, Thomas Karagiannis, Jakub Kocur, and Jacobus Van der Merwe. Echo: A reliable distributed cellular core network for hyper-scale public clouds. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 163–178, New York, NY, USA, 2018. Association for Computing Machinery.

[5] Shaddi Hasan and et al. Building flexible, Low-Cost wireless access networks with magma. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1667–1681, Boston, MA, April 2023. USENIX Association.

[6] Marius Corici, Eric Troudt, and Thomas Magedanz. An organic 6g core network architecture. In *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, pages 1–7, 2022.

[7] Hiroki Watanabe, Kunio Akashi, Keiichi Shima, Yuji Sekiya, and Katsuhiro Horiba. A Design of Stateless 5G Core Network with Procedural Processing. In *2023 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 199–204, 2023.

[8] Amazon Web Services, Inc. What is AWS Lambda?, 2023. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html.

[9] Amazon Web Services, Inc. What is AWS Simple Queue Service?, 2023. https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html.

[10] Amazon Web Services, Inc. Amazon DynamoDB, 2023. https://aws.amazon.com/pm/dynamodb/.

[11] Amazon Web Services, Inc. Amazon Relational Database Service, 2023. https://aws.amazon.com/rds/.

[12] 3GPP. Procedures for the 5G System (5GS). Technical Specification (TS) 23.502, 3rd Generation Partnership Project (3GPP), 9 2022. v17.6.0.

[13] Ali Güngör. UERANSIM v3.1.0, 2023. https://github.com/aligungr/UERANSIM.

[14] Spirent Communications plc. Landslide Core Network Testing, 2023. https://www.spirent.com/products/core-network-test-5g-lte-ims-wifi-diameter-landslide.

[15] Amazon Web Services, Inc. Amazon DynamoDB Developer Guide Read/write capacity mode, 2023. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html.