



Escola d'Enginyeria de Telecomunicació i
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Log-based monitoring, detection and automated correction of anomalies in the 5G Core

A Degree Thesis

TITULACIÓ: GRAU EN ENGINYERIA TELEMÀTICA

AUTOR: ALCALDE CÉSPEDES, JOEL FERNANDO

DIRECTOR: DOMINGO ALADRÉN, MARIA DEL CARMEN

SUPERVISOR: CATALÁN CID, MIGUEL

DATA: 8 de setembre del 2022

Abstract

This project focuses on monitoring the 5G Core through logs to identify potential problems in the operations and workflows between the different network functions and the RAN. For this purpose, it has been used an environment based on open source RAN simulators (UERANSIM and my5G-RANTester), an open source implementation of the 5G Core (Open5GS), a SDR-based radio gNB (Amarisoft) and the Grafana Loki framework as a monitoring tool. In addition, an automation solution has been developed to detect unregistered UEs and to add them to the Open5gs database based on the analysis of received logs, the creation of alerts and the development of an API.

Resumen

Este proyecto se centra en la monitorización del 5G Core a través de logs para identificar posibles problemas durante el proceso de conexión y flujos de trabajo entre las diferentes funciones de red y la RAN. Para ello, se ha utilizado un entorno basado en simuladores RAN de código abierto (UERANSIM y my5G-RANTester), una implementación de código abierto del 5G Core (Open5GS), un gNB de radio basado en SDR (Amarisoft) y el framework Grafana Loki como herramienta de monitorización. Además, se ha desarrollado una solución de automatización para detectar los UEs no registrados y añadirlos a la base de datos de Open5gs a partir del análisis de los logs recibidos, la creación de alertas y el desarrollo de una API.

Resum

Aquest projecte es centra en el monitoratge del 5G Core mitjançant logs per identificar possibles problemes durant el procés de connexió i fluxos de treball entre les diferents funcions de xarxa i la RAN. Per fer-ho, s'ha utilitzat un entorn basat en simuladors RAN de codi obert (UERANSIM i my5G-RANTester), una implementació de codi obert del 5G Core (Open5GS), un gNB de ràdio basat en SDR (Amarisoft) i el framework Grafana Loki com a eina de monitorització. A més, s'ha desenvolupat una solució d'automatització per detectar els UEs no registrats i afegir-los a la base de dades d'Open5gs a partir de l'anàlisi dels logs rebuts, la creació d>alertes i el desenvolupament d'una API.

Acknowledgements

Quisiera agradecer a mi director de tesis Miguel Catalán por ofrecerme este proyecto y ayudarme cuando lo he necesitado. También quiero agradecer a mi familia por confiar en mí en todo momento y por su apoyo cuando más lo necesitaba. Sin ellos, no habría sido posible.

List of concept

CHAPTER 1	7
1.1. Introduction	7
1.1.1. Statement of purpose	7
1.1.2. Methods and procedures	7
1.2.2. Zero-Touch Automation	8
CHAPTER 2	9
2.1. 5G Core	9
2.1.1. New architecture	9
2.1.2. Network Functions	10
2.1.3. Network Slicing	11
2.1.4. Service Based Architecture	13
2.1.5. OpenAPI	13
2.1.6. Workflows	13
CHAPTER 3	17
3.1. Core observability	17
3.1.1. Logs	17
3.1.2. Metrics	17
3.1.3. Tracing	18
3.2. Components of the analyzed use case	18
3.2.1. Open5Gs	19
3.2.2. 5G RAN	21
3.2.2.1. UERANSIM	21
3.2.2.2. my5G-RANTester	22
3.2.2.2. AMARISOFT	22
3.2.3. Grafana Loki	23
CHAPTER 4	24
4.1. Setting up the environment	24
4.1.1. Open5Gs log monitoring with Loki	25
4.1.2. my5G-RANTester log monitoring with Loki	28
4.1.3. Amarisoft log monitoring with Loki	30
4.1.4. Implementation of automatic registration of new UEs	31
4.1.4.1. Detect UE not present in the DB	31
4.1.4.2. Create an alert detecting it	32
4.1.4.3. Add it automatically to the DB	34

4.2. Experimentation	36
4.2.1. Amarisoft experiments	36
4.2.1.1. NGAP workflows	36
4.2.1.2. UE added workflows	37
4.2.2. my5G-RANTester experiments	38
4.2.2.1. NG Setup without UE	38
4.2.2.2. UE connection + UE credentials	40
4.2.2.3. Conexión UE + Slicing	43
4.2.2.4. UE connection + Session Creation	44
4.2.2.5. UE connection + NFs stopped	45
4.2.2.6. Several Slices	47
4.2.3. Experiment: automatic registration of a new user	48
CHAPTER 5	51
5.1. Conclusions	51
5.2. Future development	51
Bibliography	52
Annex	53

CHAPTER 1

1.1. Introduction

This thesis has been carried out at the i2CAT Foundation research center through an agreement with the Universitat Politècnica de Catalunya. It consists of studying the operations of the 5G Core to observe the workflows between the different network functions and the RAN part in order to detect errors and anomalies. The project has been carried out by monitoring relevant components and workflows involved in the 5G communication. To do this, an open source environment has been built that allows observing the logs of each component and forcing errors and then analyzing them to find a solution based on automation to make the network more efficient. This would considerably improve the operational expenditures (OPEX) and reduce the time expended in maintenance.

1.1.1. Statement of purpose

The main objectives of this thesis will be briefly explained below:

- Study of all the components that make up a 5G Core.
- Study of the possibilities of obtaining relevant events in both the control plane and the data plane of the 5G Core.
- Obtaining meaningful logs from the open source Open5GS and its integration with a log analysis tool called Grafana Loki.
- Workflow analysis of all 5GCore components, detection and analysis of errors in each component to find anomalies and patterns that facilitate automation.
- Design and implementation of a solution that monitors logs during operation.

1.1.2. Methods and procedures

To understand the functioning of the 5G Core, I had to learn about the role of each component and the function. I installed the open source Open5GS to implement the 5G core and analyzed the logs of each component during the connection to the RAN part using my5G-RANTester¹, as it is an open source tool that simulates the gNB and UE. To monitor 5G workflows using a log monitoring platform, it is decided to select the Grafana Loki², in which labels are applied to differentiate the logs of each component and the level of the log (info, error or warning). A label is also applied for the timestamp, so that we can know the arrival order of each log. In this way, the Core and RAN workflows can be correctly analyzed.

¹ [GitHub - my5G/my5G-RANTester: my5G-RANTester is a gNB/UE simulator for studying 3GPP standards and stressing a 5G core.](https://github.com/my5G/my5G-RANTester)

² <https://grafana.com/docs/loki/latest/>

One of the possible errors that could occur to a user when logging in has been selected. This is an error that appears when the UE is not registered in the database. The error will be detected in Loki Grafana and will trigger an alarm indicating the error and the UE's IMSI, to send it to an API based on the Python programming language, which will receive a POST with the UE's IMSI to execute an Open5Gs script and automatically add it to MongoDB. This way, the process is automated and the error does not appear again.

1.2. Network intelligence

Future networks will need to be more efficient and to address the increasing complexity and flexibility of 5G networks. Therefore, monitoring mechanisms must be used on the RAN and 5G Core side to observe what is happening during operation at all times to detect errors instantly. Zero-Touch will make it possible to automate processes without human intervention, while Artificial Intelligence will help to further enhance Zero-Touch and automation.

1.2.1. Troubleshooting

In the telecommunications industry, there is an expansion of networks in many companies around the world and this has led to the emergence of automation that replaces human labor. This new programmable technology is undergoing an evolution, including AI, so that one day networks will be fully automated, benefiting both public and private interests. As the number of network devices increases, new functionalities are generated, making the network increasingly complex. With the Fourth Industrial Revolution, it is interesting to automate as many systems as possible to avoid the need for periodic overhauls. The inability to respond instantly to network changes makes for a poor customer experience. Industry 4.0 requires more demanding connectivity due to its high productivity demands. Service providers must build platforms with integrated monitoring to ensure that wireless networks operate with specific requirements. In the future, we will see an increase in user demand, which will require new business models and for commercial reasons, the end experience needs to be improved. For the network to work well, automation and orchestration at all levels is essential, as 5G was designed to be orchestrated, due to the high complexity compared to previous generations.

1.2.2. Zero-Touch Automation

Monitoring the network allows the provider to ensure the quality of its service by constantly optimizing network resources and guaranteeing that service at all times, even when there are external changes or failures in the network. By constantly monitoring the state of the network, it will be possible to detect events that can be foreseen in advance in order to resolve them and maintain the necessary services. To save time, reduce complexity and costs during deployment, Zero Touch Automation (ZTA) has been developed as a solution to these problems. This solution allows the network to be configured and maintained automatically, without the need for any intervention.

CHAPTER 2

2.1. 5G Core

The project has focused on the new 5G architecture based on SBA (Service Based Architecture) in which each Network Function (NF) provides services to other Network Functions (NFs). This architecture model promotes decoupling of code into microservices and is widely used in cloud environments. Previous generations processed information in central servers distant from the user. With 5G technology, it will be possible to process all this information from devices very close to the user and this will allow information and metrics to be processed in real time, with very low latencies.

2.1.1. New architecture

The architecture of 5G technology has two parts: on the one hand we have the new radio (NR) and on the other hand the 5G Core. Nowadays, the majority of cell phone users use 4G, which is why it is very difficult for operators to evolve towards 5G, as it is an absolute change which involves a large investment by the operators. The 3GPP, the entity that establishes mobile telephony standards, defined the migration to pure 5G in two phases: 5G NSA and 5G SA.

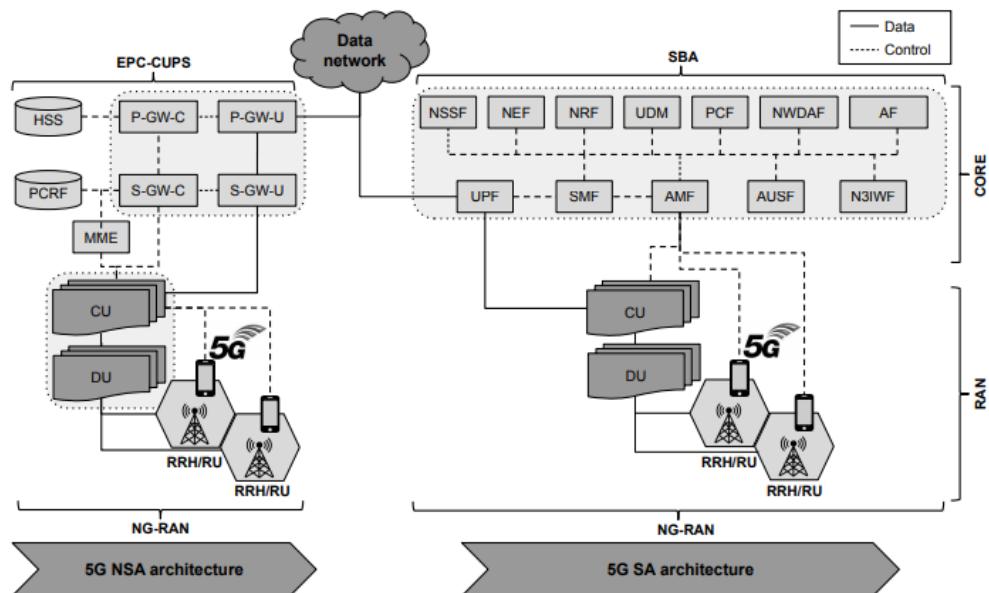


Fig 1. 5G architectures: NSA (left) and SA (right) [2].

5G NSA uses the 4G Core based on EPC (Evolved Packet Core) and CUPS (Control and User Plane Separation), which involves the separation of control and data plane. However, it uses 5G protocols for radio communication between mobiles and base stations to achieve improvements over traditional 4G. 5G SA uses 5G Core based on SBA (Service Based Architecture), which consists of using service-based interfaces between the control plane functions, while the user plane functions are connected through point-to-point links and

the radio part uses 5G NR (New Radio). 5G SA will make communications more efficient but also more expensive to deploy in large cities, as it requires a complete change in the entire radio and network infrastructure.

2.1.2. Network Functions

This section describes the main components of the 5G core and the protocols used for communication between NFs and gNodeBs.

The 5G System main objective is to provide connectivity to UEs by means of registration. 3GPP designed an architecture to manage the communication between the different components of the 5G System with the help of the NAS (Non-Access Stratum) and NGAP (NG Application Protocol) protocols.

Workflows are examined in relation to both protocols to understand the components involved. For this, the functionality of each network component needs to be studied:

- **Unified Data Manager (UDM):** It behaves as a centralized repository of data to be used during UE registration, such as storage and management of SIM identities. When there is a change in the storage data, it notifies the other NFs of these changes by identifying the subscribers. This is an important element, as it provides reliability. The UDM can store all its data in an external entity such as the UDR (Unified Data Repository).
- **Unified Data Repository (UDR):** This is the database where all data, such as subscription data and network or user policy data, is stored [5].
- **Policy Control Function (PCF):** It keeps the network unified through rules, with the aim of helping operators to control the UE's behavior in the network. In this way, it can monetise and maximize the benefits of 5G. In commercial terms, it has a great impact by reducing the cost of managing a network and also allows service providers to provide customized offerings for each and every customer.
- **User Plane Function (UPF):** It controls packet forwarding, QoS control and PDU session management. It is an evolution of the data plane function following control and user plane separation strategy (CUPS in the 4G Core). It decouples the packet gateway (PGW), user plane (PGW-u) and control plane (PGW-c) functions. This allows decentralized PGW-u packet forwarding, processing packets and aggregating traffic closer to the network, increasing bandwidth efficiency considerably. On the other hand, the PGW-c functions are provided by other NFs in the core, like the SMF.
- **Session Management Function (SMF):** It is the network function in charge of managing user sessions, connection and disconnection. The SMF assigns IP addresses for PDU sessions, as it communicates indirectly with the UE through the AMF and interfaces with the NFs

through the service-based interfaces. The SMF is responsible for choosing an appropriate UPF and this depends on the geographical location and the type of PDU. It has the function of selecting the appropriate PCF during the configuration of a PDU session and also requests information from the UDM (Unified Data Management).

- **Access and Mobility Management Function (AMF):** This network function manages the logging part, as it is a control plane function. It performs mobility functions and connection/disconnection management. When connecting to the UE, it supports encrypted signaling to confirm authentication and then registration. Its function is to identify active/inactive devices for transmission to the SMF.
- **Authentication Server Function (AUSF):** Its main function is to perform SIM-based UE authentication. The authentication process consists of a first request from the network, then it interacts with the UDM to generate authentications and finally stores KAUSF until it is removed by the AMF. The AUSF controls routing based on SUCI and SUPI and also manages a timeout to confirm authentication.
- **Network Slice Selection Function (NSSF):** 5G introduces the new concept of Network Slicing, whereby a part of the 5G network is dedicated exclusively to a specific service. Therefore, this network function helps the AMF to choose a network segment for a specific service. By dividing the network into different segments, network performance is maximized and the cost of building new networks is reduced.
- **Network Repository Function (NRF):** This network function is fundamental in 5GSA. It behaves as a central agent, where service producers advertise their capabilities. In the service-based architecture, control plane functions are configured to be registered by the NRF and this NF helps to discover other functions.
- **Network Exposure Function (NEF):** It provides an application function (AF) after exposing the services offered by the 5GCore NFs and acts as an external intermediary, retrieving information from the AMF and providing it to an external application.

2.1.3. Network Slicing

Networks must adapt to new needs, as there are many different types of services with different requirements. The single monolithic network used for multiple purposes is therefore a thing of the past. Virtualisation and SDN allow us to design logical networks (Network Slices) on a common, shared infrastructure that serves a defined business or customer purpose.

Network slicing allows the creation of many logical networks, so that they can be tailored to provide specific capabilities to meet different needs [5]. It takes on

virtualization, orchestration and network management for a better user experience by optimizing resources. There are many advantages of this network fragmentation, as it reduces time to market and allows for greater automation.

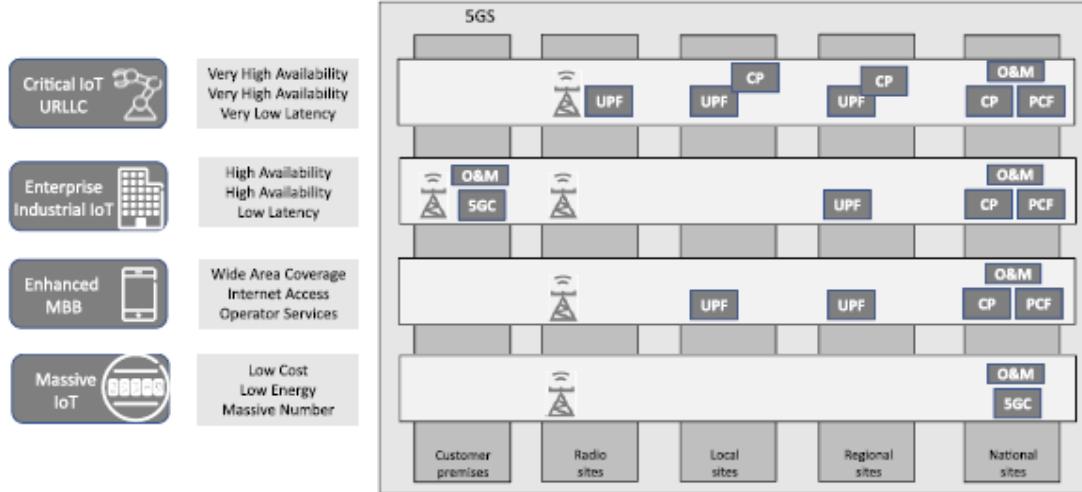


Fig. 2 Network Slice example [5].

Depending on the services, e.g. eMBB, URLLC, mIoT, and customer expectations, there may be different requirements that need to be addressed by a Network Slice. The identification of the Slice type is done through a parameter called S-NSSAI (Single Network Slice Selection Assistance Information).



Fig 3. Structure of S-NSSAI [9].

There are two fields that identify the type of Slice: the SST (Slice Service Type) which defines the type of Slice (first 8 bits) and the SD (Slice Differentiator) which is optional and serves to differentiate between different Network Slices used by different clients (24 bits).

There are SSTs already defined by the 3GPP specification that correspond to standard 5G services: eMMB (sst 1), uRLLC (sst 2), mIoT (sst 3) and V2X (sst 4). The remainder are reserved for future or customer-defined services.

For instance, a possible configuration would be `sst:1` and `sd:000001`.

2.1.4. Service Based Architecture

In the new generation architecture, services are available through the interfaces of the NFs, where each NF is accessible through an API (Application Programming Interface) and communicates via HTTP REST, a set of rules that define how web communication technologies access services through a REST (Representational State Transfer) API. The standard HTTP methods are GET (getting data), POST (sending data to a server), PUT (replacing data from a server) and DELETE (deleting data from a server). In addition, network capabilities will be easier to extend through SBI (Service Based Interface).

2.1.5. OpenAPI

HTTPS APIs use IDL (Interface Definition Language) to avoid ambiguities when making HTTP requests. 3GPP decided to use OpenAPI as the IDL for the specification of HTTP-based services. OpenAPI Initiative (OAI³) is used to describe REST APIs. The description of an API using OpenAPI includes information about the API, the resources used and the methods available on each resource (GET, POST, PUT and DELETE). OpenAPI can be said to be a specification of how to specify an API, describing it in a JSON or YAML text file. 3GPP selected YAML for its specification of service-based interfaces, as it is very easy to read and write.

2.1.6. Workflows

As mentioned in section 2.1.2, 3GPP designed an architecture to manage communication between NFs using two protocols, NAS and NGAP. The NAS data request process is initiated from the UE to the NG-RAN. NAS packets are encapsulated by the NGAP layer to be sent to the AMF. The SMF has no direct connection to the NG-RAN but communicates via the AMF.

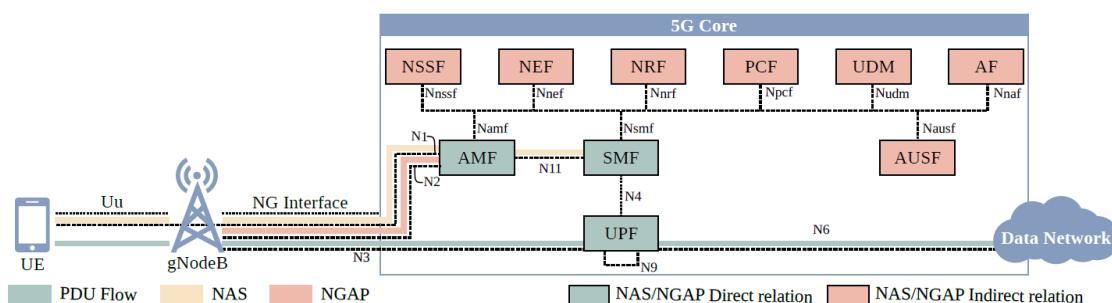


Fig. 4 5GCore SA architectural diagram [1].

To implement user data traffic procedure, the SMF connects to the UPF using the Packet Forwarding Control Protocol (PFCP) protocol. The UPF connects directly to the NG-RAN for the purpose of managing user data through the GTP-U protocol (GPRS Tunnel Protocol).

³ <https://spec.openapis.org/oas/v3.1.0>

Non-Access Stratum (NAS): Denotes the main Control Plane protocols between the UE and the core network [5]. It has the function of handling UE registration and mobility, including access control and authentication. Also establishes and maintains PDU Session connectivity and QoS for the User Plane. The NAS consists of two basic protocols to support the above functionality; the first protocol is 5GS Mobility (5GMM) and runs between the UE and the AMF to manage logs, mobility and security. The second 5GS protocol (5GSM) runs between the UE and the SMF through the AMF and supports management of PDU Session connectivity.

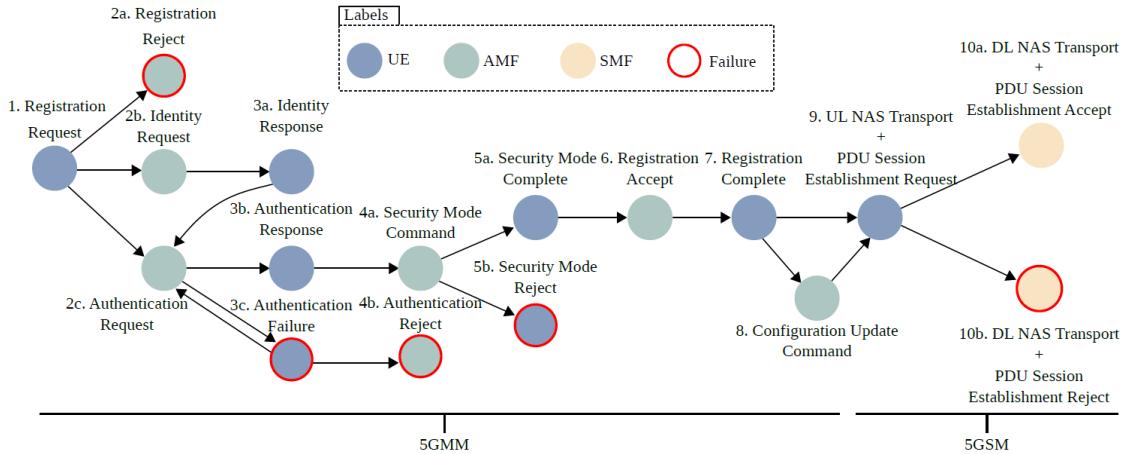


Fig. 5 NAS message flows [1].

1.- Registration: It notifies the AMF that a UE needs to register, indicates the registration status and transports information between the UE and the 5GCore network.

2.- Primary authentication and key agreement: This is a security function that consists of providing keys and the respective validations.

3.- Identification: It consists of rewriting the specific identification of the UE, e.g. SUCI, IMEI or IMEISV.

4.- Transport: It is responsible for transporting payloads between the AMF and the UE, such as NAS messages.

5.- Security Mode: A key agreement for primary authentication is established beforehand in order to subsequently establish a NAS security context between the UE and the AMF using that key.

6.- Generic UE configuration update: It is responsible for updating UE configurations with regard to access and mobility.

The first message represents the UE's registration request to the 5GC, so it needs to authenticate itself to be able to connect, in case it wants to identify

itself as a first registration in the network (SUCI) or to identify itself temporarily (GUTI).

When this request arrives at the core, the AMF processes this registration request based on three possible messages: Registration Reject (2a), Identity Request (2b) or Authentication Request (2c).

In case of any protocol error, the 5GC will request another specific Identity Response (3a). This response (3b) shall be analyzed by the core part. If the UE does not meet the requirements, the UE sends an Authentication Failure (3c), so the 5GC shall send an Authentication Reject message (4b).

The Security Mode Command (4a) carries the NAS security label selected for encryption and authentication checking. If everything works correctly, the UE responds with the Security Mode Complete (5a) message and if there are errors it will send Security Mode Reject (5b) messages.

On successful operation, the AMF will accept the registration (6) and subsequently the UE will send a message to the core stating that it has succeeded in registering (7). Finally the 5GSM functions are represented by messages (9) and (10) which consist of establishing the PDU session [1].

NG Application Protocol (NGAP): The NGAP protocol is used to interconnect gNB and 5GCore, in order to manage the processes between the RAN side and the Core. It is based on the reliable transport mechanism over STCP.

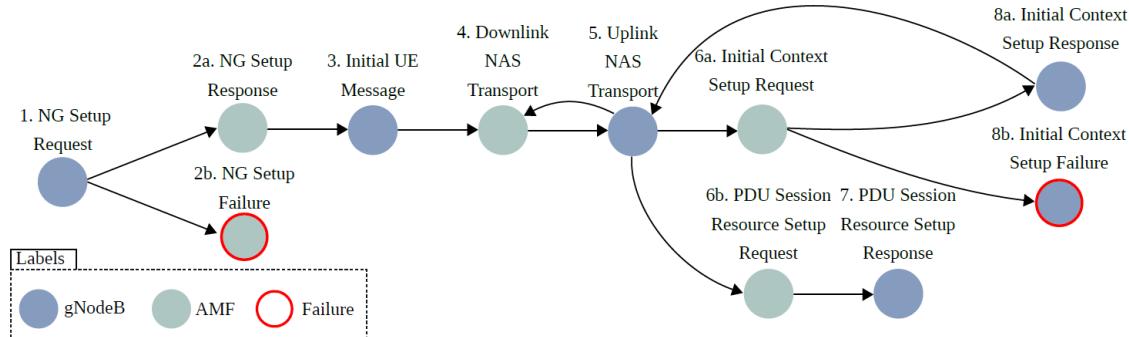


Fig. 6 NGAP message flows [1].

1.- Interface Management: Its role is to maintain the establishment and management of the NG-C interface on which PLMN (Public Land Mobile Network) based messages are carried in the AMF.

2.- Transport of NAS messages: It carries NAS messages between the NG-RAN and the AMF, encapsulated by the NGAP protocol.

3.- UE context management: At this point, information is exchanged between the UE and the NG-RAN. This communication is important to know the capabilities supported by the user and its identification.

4.- *PDU session management*: Manages the resources required for each of the associated PDU sessions between the NG_RAN and the 5GCore.

The first NGAP message is sent by the gNodeB representing the NG Setup Request. This message conveys information about the gNodeB, such as the Tracking Area (TA), PLMN, Radio Access Network (RAN) and all supported NSSAIs.

This is followed by the AMF response (2a), which includes core information, such as the AMF name, AMF region ID and the list of supported slices for the PLMN.

The exchange of information at each node is very important, as with this information the gNodeB can balance the control load between the set of AMFs. In case of incompatibility, an NG Setup Failure (2b) message shall be sent and the type of failure or incompatibility shall be indicated in this message.

The gNodeB uses the NGAP protocol to transport NAS signaling to the AMF and sends the Initial UE message (3).

The Downlink NAS Transport message (4) also transports NAS messages from the AMF to the NG-RAN and in the same way the Uplink NAS Transport message (5) will transmit the Uplink NAS Transport message (5).

Subsequently the AMF sends the Context Setup Request (6a) message to NG-RAN carrying information about the UE identification, supported network slices and enabled security keys.

The NG-RAN then processes the Initial Context Setup Request in which there may be two messages, the first one Initial Context Setup Response (8a) or the second one Initial Context Setup Failure (8b).

In the latter case the cause of the core rejection shall be notified and otherwise the Session PDU is established in which the messages PDU Session Resource Setup Request (6b), PDU Session Resource Setup Response (7), Initial Context Setup Request (6a) and Initial Context Setup Response (8a) are used.

CHAPTER 3

3.1. Core observability

As the cloudification of the 5G Core network and the widespread usage of container and microservice technologies in the 5G Core implementation increases, the network is becoming more and more complicated. To obtain optimal network performance, it is essential to observe the entire communication process between the 5G Core components. Network observability consists of three parts: logs, metrics and tracing. However, this project focuses on error identification through logs.

3.1.1. Logs

An event log is an immutable, timestamped record of discrete events that happened over time [3]. Logs are fundamental to any application, as most languages, application frameworks, and libraries contain support for logs. They are very useful and easy to implement, since adding a log line is equivalent to a print statement on the screen. That is why it is so important to keep track of what is happening at all times.

On the other hand, one disadvantage is that we have to store a large amount of logs, with the aim of filtering and processing information to later save it in a database. When it is possible to manage this large amount of logs, it is interesting to use tools that act as a search engine for a specific type of log. In our use case, we use a tool to handle 5GCore logs, since the 5g components analyzed in this project are constantly reporting logs, so that we can have evidence of the errors that occurred during the operation.

3.1.2. Metrics

Metrics are a numerical representation of data measured over a time interval [3]. With the help of metrics it is possible to predict the behavior of a system, to analyze performance and to detect anomalies from previously obtained values. Metrics can be monitored in real time via dashboards and are usually defined as time series: streams of timestamped values belonging to the same metric and the same set of labeled dimensions. During the operation of the core, metrics on the number of connected/disconnected users, available antennas, packet counter, etc. can be obtained.

Metrics are interesting to obtain the performance of a core (users, pdu sessions, ue performance, NF status...) but in this project we focus on logs. Currently Open5Gs is already advancing in the exportation of metrics to Prometheus⁴ with the objective of visualizing or exporting them to other systems such as Grafana. So far, only the Open5Gs SMF component supports the exportation of metrics,

⁴

<https://www.google.com/url?q=https://open5gs.org/open5gs/docs/tutorial/04-metrics-prometheus&sas=D&source=docs&ust=1662122057484784&usg=AOvVaw06B3zmNuCGez57AkYjvpBJ>

but soon other components will do so as well. This project focuses on obtaining metrics from the logs themselves to evaluate an anomaly and create alerts.

3.1.3. Tracing

A trace is a representation of a series of causally related distributed events that encode the end-to-end request flow through a distributed system [3] and provides visibility of the entire path taken by a request. It is just as important as logs and metrics, as they form the pillars of observability. The importance of the trace comes from the fact that it makes the engineer's job easier, as it helps to understand the different services involved in the path of a request, as well as monitoring performance and detecting anomalies.

While it is true that tracing tends to be very useful in microservices environments, it is ideal for SBA-based 5GCore. However, tracing is useful for any complex application where it competes for network resources. On the other hand, it should be noted that trace is difficult to adapt to an existing infrastructure, as for trace to be effective, each component of the route must be modified to propagate information.

In open source environments, it is difficult to implement tracing because a large number of developers program additional instrumentation and this makes it even more difficult. For this reason, the possibility of implementing it was ruled out due to the complexity of the Open5Gs code.

3.2. Components of the analyzed use case

This thesis aims to demonstrate the importance of log observability during 5G operation and its applicability to anomaly detection and automated correction.

The objectives of this section are defined as follows:

- To have a monitored environment.
- To analyze the workflows between the RAN and Core part during different procedures.
- To analyze logs and detect anomalies during the different procedures.
- To automate corrections according to obtained logs

On the other hand, it describes the components considered in the experiments:

- Core observability: Use of Grafana, Grafana Loki and Promtail.
- 5G Core: Open5Gs open-source implementation.
- 5G RAN: Use of two simulators (UERANSIM and my5G-RANTester) and a real 5G RAN (Amarisoft and 5G Modem).
- Developed API: An API is developed that gets alerts from Grafana and interacts with Open5GS.

These points are discussed in more detail in chapter 4. The tools used to configure our usage environment and their justification will be explained.

3.2.1. Open5Gs

The 5G and LTE network is deployed with Open5Gs, an open source C-language implementation for 4G and 5G cores. It is open source software, which means that you can contribute to the code. As it is open source, you can do business, depending on the license there are more or less restrictions⁵ (GNU Affero General Public License v3.0 in case of Open5Gs). Open5Gs uses a service-based architecture (SBI), as the control plane functions are configured to register to the NRF and then this component helps to discover the other functions.

On the other hand, since it uses YAML based configurations, simple and flexible configurations can be made. The operating system used is Linux and during deployment we get different levels of logs. This tool has a hybrid implementation of EPC (4G) and 5G Core, where some elements are shared. The elements are designed as VNFs (Virtual Network Functions) and use CUPS to separate the control plane from the user plane [7].

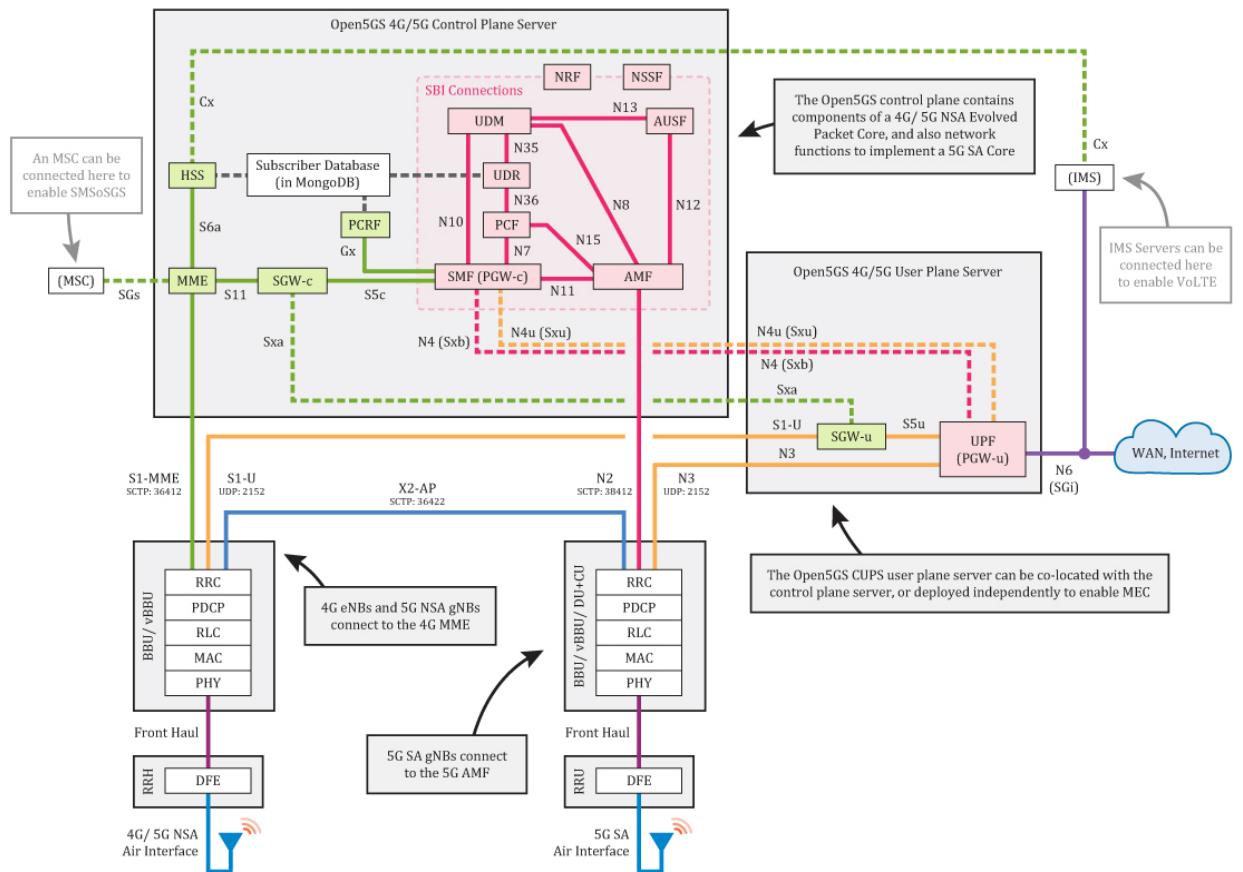


Fig 7. Architecture of Open5Gs [7].

⁵ <https://github.com/open5gs/open5gs/blob/main/LICENSE>

Fig 7 shows the 4G components in green and the 5G components in pink. In this case, the user plane is simpler, as it has only one function (UPF), to transport UE packets between the gNB and the external WAN. Next figures show the Open5Gs GUI (Graphical User Interface) to manage the addition, modification and deletion of users, where the MongoDB database to store UEs is used.

The screenshot shows the 'Add User' interface of the Open5Gs GUI. It includes the following fields:

- IMSI***: 001010000000001
- Subscriber Key (K)***: 465B5CE8B199B49FAA5F0A2EE238A6BC
- Authentication Management Field (AMF)***: 8000
- USIM Type**: OPc
- Operator Key (OPc/OP)***: EBED289DEBA952E4283B54E88E6183CA
- UE-AMBR Downlink***: 1 Gbps
- UE-AMBR Uplink***: 1 Gbps
- Slice Configurations**:
 - SST***: SD
 - SD**: Radio 1 (selected)
 - Default S-NSSAI**: checked

Fig 8. Parameters for adding a user in the Open5Gs GUI.

Fig 8 shows the mandatory data to add a user, which is the *IMSI (International Mobile Subscriber Identity)*, *K (Subscriber Key)*, *OPC (Operator Key)*, *AMF (Authentication Management Field)* and *SST (Slice Service Type)*. These are mandatory parameters as they are responsible for the identification of a specific user, its credentials and the allowed slices.



Fig 9. Open5Gs GUI.

3.2.2. 5G RAN

In the RAN part we used different simulation tools: first UERANSIM, then my5G-RANTester and finally Amarisoft as real 5G RAN. After having analyzed the actual behavior between radio and core, we decided to analyze this workflow between UE, radio and CORE using the previously mentioned open source tools.

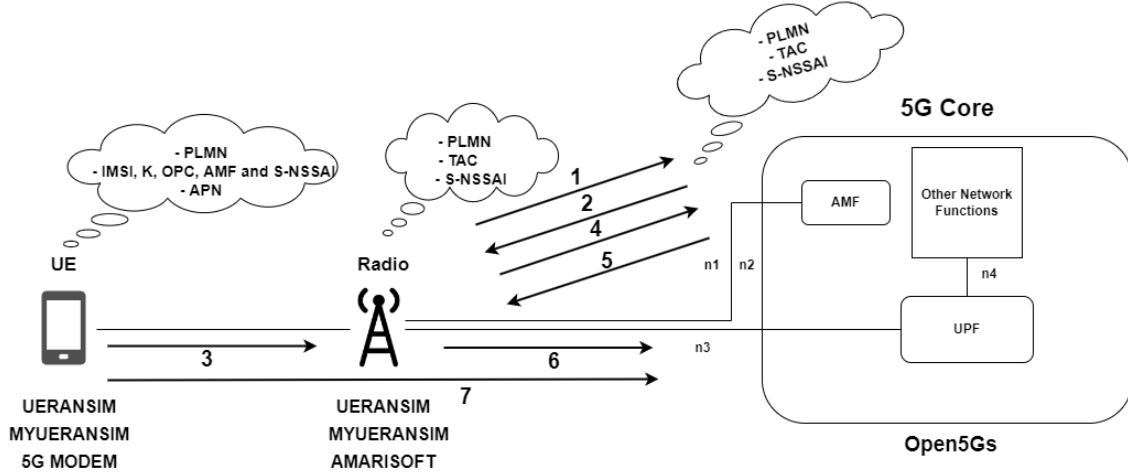


Fig 10. My environment diagram.

First, the gNB will send some parameters to the AMF (1), and then the AMF will reply to the gNB (2); this is a NG Setup process in order to check if the Core can serve that gNB. If they are correct (3), the gNB will start radiating the PLMN and the UE will be able to connect to the gNB.

The next step is to send information (4) from the UE to the AMF to check if the user can connect (5), at this point the PDU session is created. When the user is already registered, the GTP tunnel is created between the gNB and the UPF (5) and finally the UE sends data through the gNB to the UPF (6). In the following, each 5G RAN tool will be explained with the one used to test.

3.2.2.1. UERANSIM

This is a very useful open source tool, as it simulates a 5G mobile phone and a base station, using the C++ programming language. The aim of this simulator is to test the 5G core network and study the workflows.

It contains three interfaces in the UE/RAN perspective. During the realization of the project, connectivity tests were performed with Open5Gs but the experimentation part was not carried out.

3.2.2.2. my5G-RANTester

It is a tool that behaves very similar to UERANSIM. It follows the 3GPP standard and is very interesting because different workloads can be generated to see how the core works and in case of error, it will notify it based on 3GPP standards.

Unlike UERANSIM, it is very scalable, as it is able to mimic the behavior of many UEs and gNBs. It is also capable of testing AMF responses per second and UE log latency.

3.2.2.2. AMARISOFT

Amarisoft is a closed source commercial solution based on licenses, and provides the industry with hardware able to implement a cellular RAN (eNB and gNB). Using amarisoft we can get the 4G/5G network in a lab for testing using a Software Defined Radio (SDR) based system, where each SDR has two transmit, two receive and one GPS (optional) interfaces. An SDR can go up to 2x2 Multiple-Input Multiple-Output (MIMO).

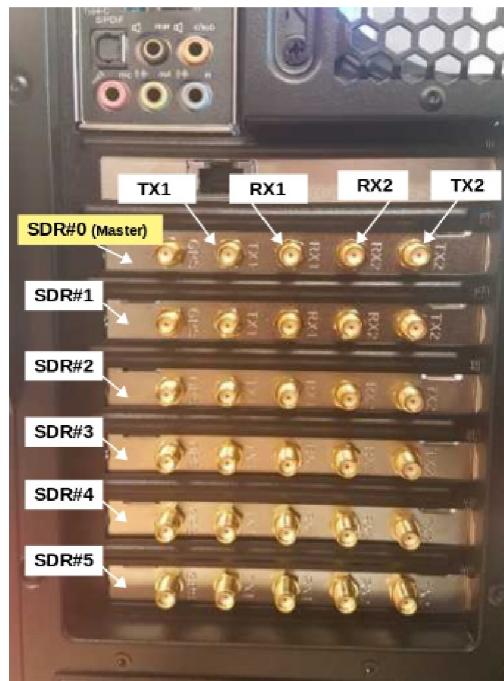


Fig 11. Amarisoft interfaces.

The current hardware in the laboratory is limited to 56MHz, but versions up to 100 MHz are available. Each 4x4 MIMO cell needs to use 2 SDRs. On the other hand, RF (radio frequency waves) combiners can be used to minimize the number of antennas as the number of cells increases.

The interfaces used by Amarisoft are ethernet and are oriented towards the Core connection. It does not use a display interface, so access is via SSH with the Linux-based operating system and the configuration files are JSON-based.

Amarisoft has its graphical interface in which we can see in real time the messages of the different layers between the radio, Core and UE. The GUI also displays cell usage statistics and radio status. The most important part of this project is the logs generated during the connection, as these messages are saved in text files for later monitoring between the Amarisoft logs and the Core with Loki Grafana.

3.2.3. Grafana Loki

Grafana Loki is a new open source product on the market based on Prometheus (a tool used to obtain metrics in any environment). Loki uses the same tag system as Prometheus as it is a log aggregation system designed to store and query logs from any application or infrastructure. It is ideal for this project as it is very scalable and designed to be very cost effective and easy to use. This tool is programmed in the GO language to improve performance, as this new language is designed to be very fast because it is interested in following the logs in real time to see logs as they enter the system.

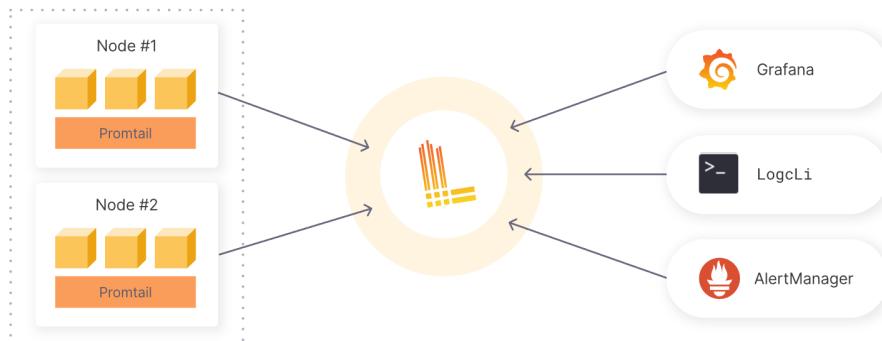


Fig 12. Functioning of Loki Grafana [6].

The first step is to extract any kind of logs with Promtail in order to tag, transform and filter logs from any application, that's why it was installed in the monitored virtual machine itself. The second step is to install Loki in the Cloud to store the logs in Loki, as the entries are grouped in sequences and indexed with tags. This not only reduces costs, but also makes it possible to query the logs within milliseconds after Loki receives them. The third step is to use LogQL directly in Grafana or LogCLI through the command line to scan the logs and finally Grafana will create the alert and send them through Grafana's own AlertManager.

CHAPTER 4

4.1. Setting up the environment

This chapter will show all the work done for the installation and configuration of the environment. All the configuration of the Core using Open5Gs and the configuration of the RAN part, both UERANSIM, my5G-RANTester and Amarisoft.

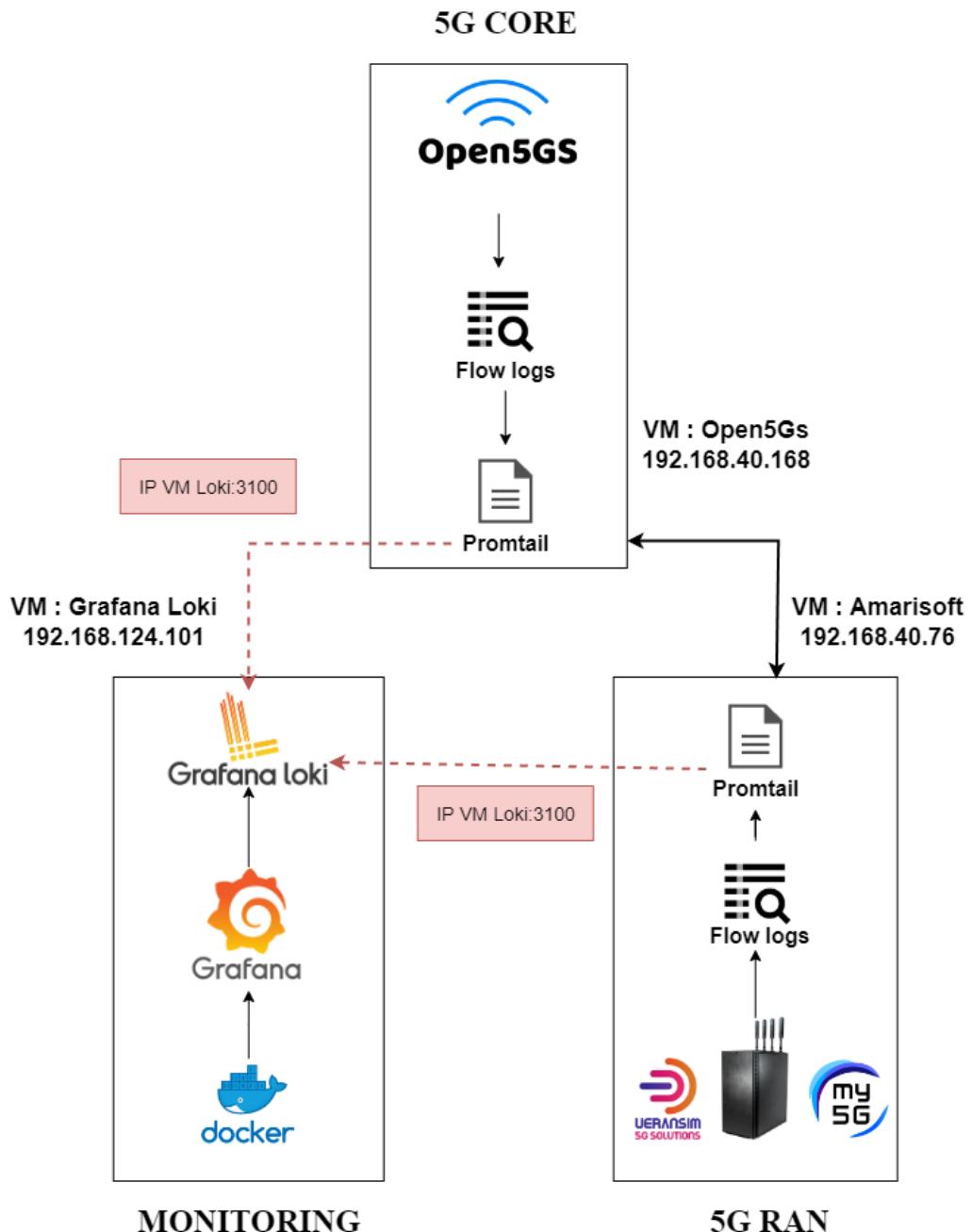


Fig 13. Deployment diagram.

Both simulators generate logs during deployment and these logs will be manipulated by Promtail, since it is an agent that sends the content to Grafana Loki to be monitored.

The use of Promtail was very important, because it attaches tags to the logs. Open5GS and Amarisoft are directly connected through the IP address assigned to each virtual machine.

First of all, we must install and configure Loki because it is our primary tool to monitor the Core logs and the RAN part. To do this, Loki is installed using git on the VM with the IP 192.168.40.76. The Loki VM is accessed via ssh with a keypair and Loki is installed.

```
git clone https://github.com/grafana/loki.git
```

Then in the directory `/home/ubuntu/loki/production` contains the docker compose necessary to run the program, so we proceed to run it.

```
sudo docker-compose pull  
sudo docker-compose up
```

When installing loki from github, by default we get three images, Grafana, Loki and Promtail. In my use case only the Grafana and Loki images are needed, as the other Core and RAN VMs will use this third promtail image.

The docker compose yaml file is modified to obtain only the images of Loki and Grafana, on the other hand it is configured so that Grafana runs on port 4000, as shown in Fig 13, and we proceed to enter the GUI to add a datasource with the url <http://loki:3100>, with the aim that both the Core and the RAN point to the same url to monitor.

Once the Loki Grafana environment is configured, promtail is installed using docker in the Core and RAN to be able to manipulate the logs and monitor from the Loki GUI.

4.1.1. Open5Gs log monitoring with Loki

Open5Gs is installed via git on the VM with IP 192.168.40.168. However, some experiments also installed an Open5Gs on the same virtual machine as 5G RAN and Loki. We access the Open5Gs VM through ssh with a keypair and install Open5Gs and the next step is to install Loki to get the promtail image.

For this project, Open5Gs version v2.4.9 and Loki Grafana version v2.6.1 have been used.

```
git clone https://github.com/open5gs/open5gs.git  
git clone https://github.com/grafana/loki.git
```

Network Function	Directory	Configuration
AMF	/etc/open5gs/amf.yaml	<pre> plmn_support: - plmn_id: mcc: 001 mnc: 03 s_nssai: - sst: 1 </pre>
SMF	/etc/open5gs/smf.yaml	<pre> info: - s_nssai: - sst: 1 dnn: - internet - sst: 2 dnn: - internet </pre>
NSSF	/etc/open5gs/nssf.yaml	<pre> nssf: sbi: - addr: 127.0.0.14 port: 7777 nsi: - addr: 127.0.0.10 port: 7777 s_nssai: sst: 1 - addr: 127.0.0.10 port: 7777 s_nssai: sst: 2 </pre>

Table 1. Open5Gs configuration files.

After installation, the AMF, SMF and NSSF are modified in the directory `/etc/open5gs/`, as shown in Table 1. The other parameters are already set by default and do not need to be manipulated on this occasion. Sometimes (e.g. Amarisoft) it was necessary to modify the AMF and UPF files to configure the interfaces connected to Amarisoft.

In the AMF a `plmn_id` of `mcc:001` and `mnc:03` is configured, on the other hand an `sst:1` is configured, this value must match the gNodeB that will be configured later. In the SMF and NSSF we define the supported `sst` with the corresponding name (`dnn`), by default it is `internet`.

When configured, the logs exchanged by the NFs during the operation will be seen, so we only need the promtail docker image to manipulate the Open5Gs logs, so we decided to install only this image and the other two images (Loki and Grafana) will be removed, as by default the three images come with it.

In the directory `/loki/production/simple_scalable` is the yaml configuration file called `promtail-config.yaml`, which will be modified.

```

- job_name: open5gs
  static_configs:
    - targets:
      - localhost
    labels:
      job: open5gs
      __path__: /var/log/open5gs/*.log

```

Fig 14. Creating a job for Open5Gs (Annex 2).

As mentioned in section 3.2.3, Loki Grafana is based on Prometheus, which defines a set of instances called job, so we define a *job_name* inside the configuration file called Open5Gs and indicate the *__path__* where all the Open5Gs logs are located. This is how the Core information is injected into Loki Grafana. Promtail discovers locations of log files and extracts labels from them through the *scrape_configs* section in the config YAML.

The *pipeline* is also used to transform a log line, its tags and its timestamp. The *pipeline* starts with a match stage, in which the previously declared *job_name* will be selected, in order to extract the log data and split it into different tags for easier monitoring. The regex expression is used to capture part of the log and distinguish it by labels.

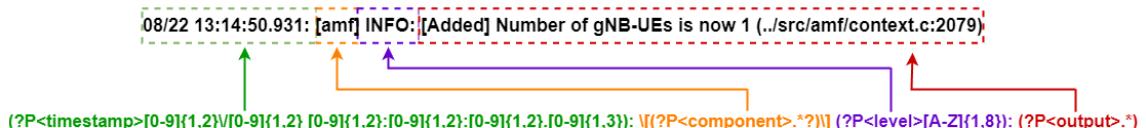


Fig 15. Open5Gs regex expression.

Fig 15 shows an example of how to structure an AMF log from Open5Gs and its respective regex expression to divide it into three labels, first the timestamp is captured (green), then the component (orange) and finally the log level (purple), everything that follows will be the log output (red).

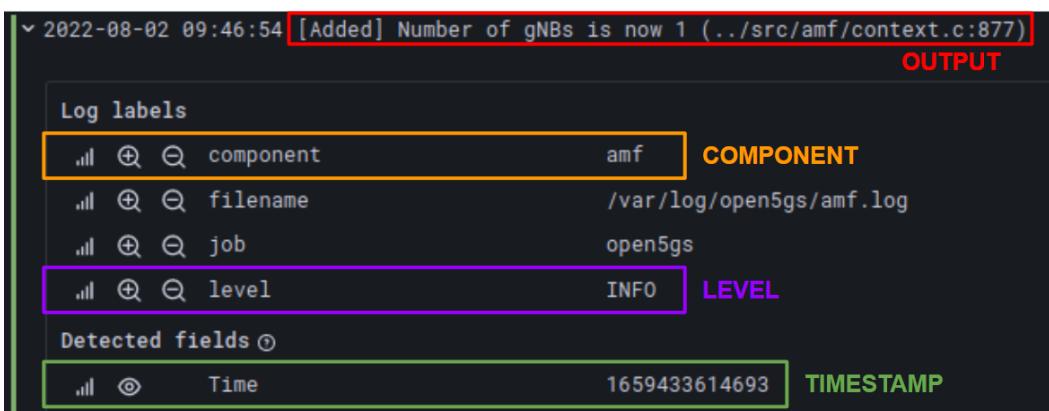


Fig 16. AMF log in Loki.

After configuring the regex expression in the promtail (Fig 15), we obtain the distribution of labels in Loki divided into component, log level and timestamp as shown in Fig 16. This way, the Open5Gs logs are fully characterized and it will be much easier to monitor the 5GCore and in case an error appears during the

operation, it will be easier to find it, as it can be filtered by log level (e.g. ERROR) and by component. By having Open5Gs monitored, we can observe the amount of logs that appear over a period of time and this is very useful to get an idea of the state of the network through the logs. In the promtail, all Open5Gs NFs were configured in debug mode for the logs, in order to obtain the maximum information, so some logs were filtered via promtail in order not to saturate Loki by using drop expression.

4.1.2. my5G-RANTester log monitoring with Loki

my5G-RANTester version v2.4.9 is used and installed via git on the virtual machine 192.168.40.76, as this machine will be used to simulate the RAN part. Loki is then installed to get the promtail image as well as Open5Gs.

```
git clone https://github.com/my5G/my5G-RANTester.git
git clone https://github.com/grafana/loki.git
```

To configure the yaml file, go to the `/my5G-RANTester/config` directory and configure it with the same parameters as Open5gs so that the connection between the two is made correctly.

Configuration
<pre>gnodeb: controlif: ip: "127.0.0.1" port: 9487 dataif: ip: "127.0.0.1" port: 2152 plmnlist: mcc: "001" mnc: "01" tac: "000001" gnbid: "000001" slicesupportlist: sst: "01" sst: "02"</pre>
<pre>ue: msin: "0000000001" key: "465B5CE8B199B49FAA5F0A2EE238A6BC" opc: "E8ED289DEBA952E4283B54E88E6183CA" amf: "8000" sqn: "0000000" dnn: "internet" hplmn: mcc: "001" mnc: "03" snssai: sst: 01</pre>

Table 2. my5G-RANTester configuration.

As shown in Table 2, both gNodeB and UE have the same mcc and mnc as Open5Gs. These are example values to show the configuration, but may change depending on the need of each service.

```
- job_name: my5G_RAN_tester
  static_configs:
    - targets:
      - localhost
    labels:
      job: my5G_RAN_tester
      __path__: /var/log/my5G_RAN_tester/logs_my5G_RAN_tester.txt
```

Fig 17. Creating a job for my5G-RANTester (Annex 3).

This virtual machine will also have a promtail with a *job_name* called *my5G_RAN_tester*. When running my5G-RANTester, the logs are printed on screen and for my use case it is necessary to have these logs stored in a text file in order to be able to observe them in Loki Grafana.

It is decided to use ">>" in Linux to redirect all logs to a directory where my text file defined in the *job_name* is located as follows:

```
./app ue >> /var/log/my5G_RAN_tester/logs_my5G_RAN_tester.txt
```

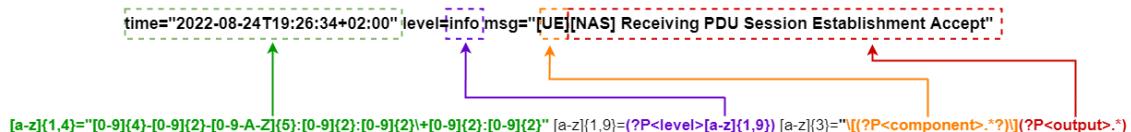


Fig 18. my5G-RANTester regex expression (Annex 2).

my5G-RANTester records have a different structure than Open5Gs records, so a different regex expression will be applied.

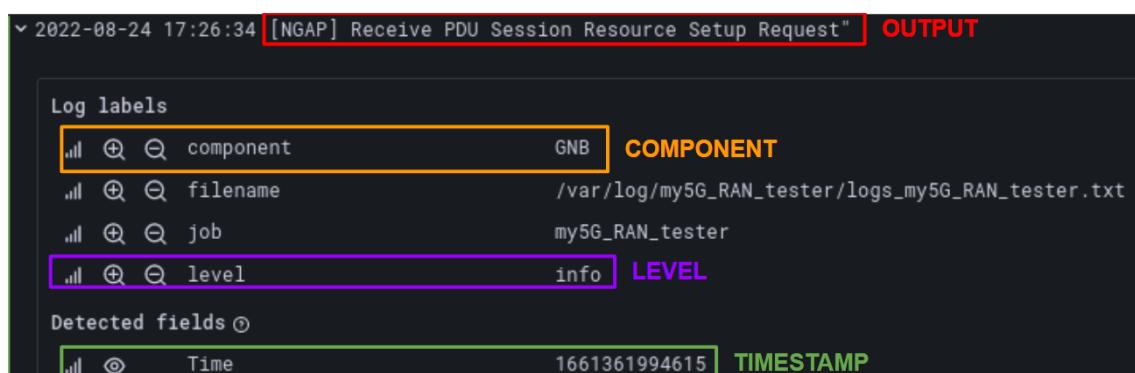


Fig 19. my5G-RANTester log in Loki.

The three labels are distinguished and this is how the RAN part can be monitored.

4.1.3. Amarisoft log monitoring with Loki

The Amarisoft machine is accessed via ssh and promtail is installed to apply a suitable regex expression and Loki is installed to obtain the promtail image.

```
git clone https://github.com/grafana/loki.git
```

To configure the relevant parameters to make the connection to the 5GCore, go to the `/root/enb/config/fp_test_5g_sa/nr-cells.cfg` directory and modify the json file.

Configuration
<pre>"plmn_list": [{ "plmn": "00103", "reserved": false, "tac": 67, "nssai": [{"sst":1, "sd":000000}] }],</pre>

Table 3. Amarisoft configuration.

As shown in Table 3, we configure the `"plmn" : "00103"` exactly the same as Open5Gs, which would be the equivalent of mcc and mnc.

```
- job_name: amarisoft
  static_configs:
    - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /tmp/5g_sa.log
```

Fig 20. Creating a job for Amarisoft (Annex 1).

A new job called amarisoft needs to be created in order to apply the pipeline and generate a suitable regex expression for Amarisoft logs.



Fig 21. Amarisoft regex expression (Annex 1).

In this way we have the amarisoft logs labeled with the timestamp and with their respective components observed in Loki, similar to Fig 19 of my5G-RANTester.

In the Amarisoft promtail, some logs were also filtered via drop expression in order not to saturate Loki and to get the relevant logs to analyze the workflows. On the other hand, a timestamp was also added to be able to observe the time of appearance of logs and compare it with the Open5Gs logs and observe the order of arrival of each log.

4.1.4. Implementation of automatic registration of new UEs

The implementation of user management is part of Zero-Touch, as the aim is to automate the process of adding a new user that is not previously registered in the Open5Gs database. This implementation could be very interesting in a private 5G network scenario, where the customer does not want to be given access to the core dashboard, but the facility to add new UEs to their network without the need for maintenance from the radio and core provider.

To do this, the IMSI of the unregistered user must first be detected by viewing the logs during the connection. Once the error log containing the IMSI is detected, an alarm is created in Loki Grafana to send a post webhook message to the python-based flask endpoint. This endpoint receives that IMSI and applies the scripts defined by Open5Gs to automatically add an IMSI to MongoDB.

4.1.4.1. Detect UE not present in the DB

When studying the behavior of each component of the 5GCore, it is known that the UDR is in charge of storing all the IMSIs of the users registered in the database, therefore, it is the one that warns through a log that this user is not in MongoDB with the message "*Cannot find IMSI in DB*". Therefore, it is decided to create a new job inside the promtail file of the Open5gs called *job_name: imsi* with the address of only the UDR logs.

```
- job_name: imsi
  static_configs:
    - targets:
        - localhost
      labels:
        job: imsi
      __path__: /var/log/open5gs/udr.log
```

Fig 22. Job to detect IMSIs (Annex 2).

In this new job, it is only necessary to capture the IMSI and then pass it as a parameter to the webhook to automate the user's aggregation.

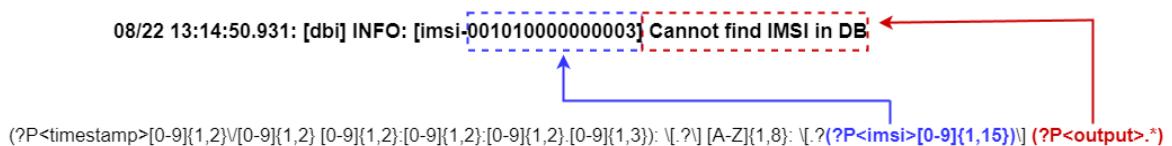


Fig 23. Regex expression in UDR to only capture IMSI.

After having captured the IMSI that is not yet registered in the database, we proceed to search for the number of times that log appears during a period of time in the Loki Grafana search engine.



Fig 24. Log browser of Loki.

As previously mentioned, it was decided to create a new job in the Open5Gs promtail to capture the IMSI using regex expression, in order to observe the captured IMSI in the Loki Log Browser.

Therefore, in Fig 24 shows “`count_over_time({job="imsi"} |= "Cannot find IMSI in DB"[20s])`” which shows the graph of how many times this log appears during 20s, selecting the previously created job to capture only the IMSI.

The three promtail values are also shown: `filename`, `imsi` and `job`. However, we are only interested in the IMSI value to automate the process of adding an unregistered user.

4.1.4.2. Create an alert detecting it

Next, we decide to create an alarm to detect the log seen in Fig 23. To do this, we go to *Alerting* → *Alert Rules* in Loki Grafana to define our rules that will trigger the alarm.

The figure shows the Loki Alert Rules interface. At the top, there is a search bar with the text "now-20s to now" and a "Run queries" button. Below the search bar is a "QUERY" section containing the query "count_over_time({job='imsi'} |= \"Cannot find IMSI in DB\"[20s])". In the main area, there is a section labeled "B (Expression)" with a red border around it. This section contains an "EXPRESSION" tab and an "Operation" dropdown set to "Classic condition". Below the operation dropdown is a "Conditions" row with the following fields: "WHEN" (set to "last()"), "OF" (set to "A"), "IS ABOVE" (set to "0"), and a delete icon. There is also a "+" button to add more conditions.

Fig 25. Alarm conditions.

It is decided to create a query and an expression, since in the query we define the log that we are interested in detecting and the times of occurrence (`count_over_time`) and in the expression it is decided to activate the alarm when

a number of logs greater than 0 appears, therefore, if only one "Cannot find IMSI in DB" log appears, the alarm will be activated (IS ABOVE = 0).

In a real case this number could be increased to trigger the alarm when it happens more than a certain number of times and thus avoid false positives. In this way we have already defined the conditions for the alarm to be triggered, however, it is necessary to define more parameters. The time interval in which the alarm is evaluated is defined with a parameter.

Evaluate every 10s for 20s

Fig 26. Alert evaluation behavior.

To make the alarm as fast as possible, it is decided to evaluate every 10s for 20s. This evaluation interval applies to each rule within a group. Therefore, it is necessary to create a group of alarms.

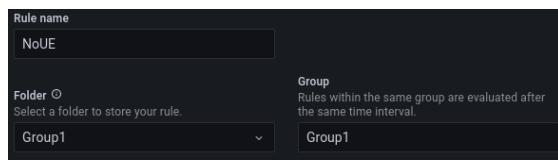


Fig 27. Add details for my alert.

Then the alarm identification details are added, e.g. the name of the rule, the folder where the rule is stored and the group in which it will be evaluated after the same time interval.

Summary {{ \$value }}

Fig 28. Aggregation of extra information.

In the Summary section a comment on the alarm status is left, however, it was decided to pass the promtail values previously defined using "{{ \$value }}", in which the values of filename, imsi and job will be passed.

State	Labels	Created
Alerting	alertname=NoUE, component=component, error=noIMSI	2022-08-26 19:20:53
Value	1e+00	
Summary	[var='B0 metric=(filename=/var/log/open5gs/udr.log, imsi='00101000000004, job=imsi)' labels=(filename=/var/log/open5gs/udr.log, imsi=00101000000004, job=imsi) value=1]	

Fig 29. Alarm in firing state.

Fig 29 shows the alarm status, in the Summary parameter the values are defined with "{{ \$value }}", in which the IMSI of the unregistered user appears. In

this way the alarm is configured and the next step is to connect it to the webhook.

4.1.4.3. Add it automatically to the DB

When the alert is already created, a Webhook needs to be created in *Alerting* → *Contact Point* to send all this information to a Python-based POST method endpoint.

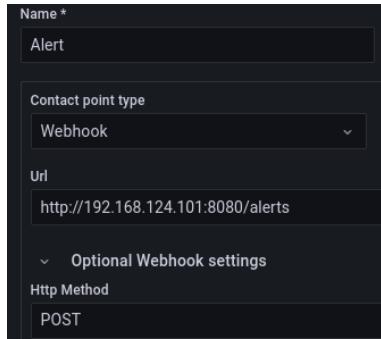


Fig 30. Contact point in Loki.

A new contact point called Alert of type Webhook is created with the URL defined in the python code. Next, an endpoint based on the python programming language is created. This Python flask will receive the previously configured alert when it is in firing state. First, the necessary libraries must be imported to be able to program the endpoint.

```
from urllib import response
from flask import Flask, request, jsonify
from pymongo import MongoClient
import datetime as DT
import os
```

Fig 31. Python libraries.

The urllib library is imported, as it is the library in charge of compiling several modules to work with URLs, also the Flask framework is imported to be able to create web applications in a very fast way. Pymongo is a python distribution that contains tools to work with MongoDB, as it was decided to store all the IMSIs not registered in a MongoDB table, therefore, datetime is also imported because the IMSI will be added and in another column the timestamp.

Finally, an OS is imported in order to run Open5Gs scripts to add IMSIs to the database without using a command line, because this way the whole process would be fully automated.

```
MONGO_URI = 'mongodb://localhost'
client = MongoClient(MONGO_URI)
db = client['newUE']
collection = db['imsi']
```

Fig 32. Creating a database.

It is necessary to create a new database called "newUE" with a collection called "imsi" and define the url of the database.

```
def minutes(timestamp=None):
    if timestamp is None:
        now = DT.datetime.now()
        timestamp = now.timestamp()
    return timestamp//60
```

Fig 33. Minute function.

A function is defined that returns the time in minutes, used for storing IMISIs with their respective timestamp.

```
app = Flask(__name__)
@app.route('/alerts', methods=['POST'])
def main():
    min = minutes()
    payload = request.get_json(force=True)
    alerts = []
    alerts = payload['alerts']
    for x in alerts:
        annotations=[]
        annotations = x['annotations']
        try:
            summary = []
            summary = annotations['summary']
            valor = summary.split("imsi=")[2]
            imsi = valor.split(',')[0]
            find = collection.find()
            results = list(find)
            imsis = dict()
            for y in collection.find():
                imsis[y['imsi']] = y['date']
            if (imsi not in imsis.keys()):
                print ("New IMSI "+imsi+" added")
                collection.insert_one({"imsi":imsi, "date" : min})
                os.system('bash ./script.sh add '+imsi+ ' 465B5CE8B199B49FAA5F0A2EE238A6BC E8ED289DEBA952E4283B54E88E6183CA')
                exit()
            elif (imsis[imsi] + 1 < min):
                collection.delete_one({"imsi":imsi})
                print("IMSI "+imsi+" repeated but added to DB")
                collection.insert_one({"imsi":imsi, "date" : min})
                os.system('bash ./script.sh add '+imsi+ ' 465B5CE8B199B49FAA5F0A2EE238A6BC E8ED289DEBA952E4283B54E88E6183CA')
                exit()
            else:
                print("Ignored "+imsi+" IMSI")
        except:
            print("")
    return 'success', 200
```

Fig 34. Main function.

Finally, it is decided to create a code based on Flask, as shown in Fig 34, which performs the function of an endpoint with the POST method in which a vector is created that will obtain the value of the alert created in the previous section.

The aim of this code is:

- If the IMSI is not in the newly created database, it adds it to the 5G Core via an Open5Gs script.
- If the IMSI is in our database and more than one minute has passed since it was added, it updates the database and puts that IMSI back into the database. This is needed in case the two databases are not synchronized, for instance if the Open5Gs database has been externally modified.
- Otherwise, the alert will be ignored.

This would result in an efficient and fast automation of the management of new users.

4.2. Experimentation

This section will show all the experiments carried out once the environment has been correctly configured. Once the RAN part and the Core have been monitored, it is interesting to be able to carry out different tests using my5G-RANTester and Amarisoft. In the case of my5G-RANTester, numerous tests of all kinds were carried out, while with Amarisoft the connection with Open5gs was checked and certain logs were forced to appear in order to analyze them later.

In the case of Amarisoft, several experiments were carried out to analyze the functionality of the whole system, in selected cases of correct and incorrect workflows. However, my5G-RANTester was prioritized for testing because of its lower complexity during experimentation. On the other hand, UERANSIM was not tested, as it was not necessary if we already use the other open source tool.

4.2.1. Amarisoft experiments

This section will show two cases to demonstrate the experimentation between Amarisoft and Open5Gs.

4.2.1.1. NGAP workflows

The screenshot shows a terminal window displaying log messages from two processes: NGAP (represented by red text) and AMF (represented by green text). The log messages are interleaved, indicating simultaneous operations between the two components. The NGAP process is primarily responsible for the initial setup and configuration exchange, while the AMF process handles the connection establishment and related parameters.

```
> 2022-04-26 12:59:36 FROM - - 172.16.2.133:38412 NG setup response | NGAP
> 2022-04-26 12:59:36 IP[172.16.2.144] RAN_ID[12] (.../src/amf/ngap-path.c:64)
> 2022-04-26 12:59:36 NGSetupResponse (.../src/amf/ngap-build.c:36)
> 2022-04-26 12:59:36 NG-Setup response (.../src/amf/ngap-path.c:244)
> 2022-04-26 12:59:36 S_NSSAI[SST:1 SD:0x0] (.../src/amf/ngap-handler.c:76)
> 2022-04-26 12:59:36 PLMN_ID[MCC:1 MNC:3] (.../src/amf/ngap-handler.c:72)
> 2022-04-26 12:59:36 SERVED_TAI_INDEX[0] (.../src/amf/ngap-handler.c:46)
> 2022-04-26 12:59:36 PLMN_ID[MCC:1 MNC:3] (.../src/amf/ngap-handler.c:45)
> 2022-04-26 12:59:36 TAC[67] (.../src/amf/ngap-handler.c:40)
> 2022-04-26 12:59:36 S_NSSAI[SST:1 SD:0x0] (.../src/amf/ngap-handler.c:271)
> 2022-04-26 12:59:36 PLMN_ID[MCC:1 MNC:3] (.../src/amf/ngap-handler.c:234)
> 2022-04-26 12:59:36 TAC[67] (.../src/amf/ngap-handler.c:202)
> 2022-04-26 12:59:36 PagingDRX[2] (.../src/amf/ngap-handler.c:179)
> 2022-04-26 12:59:36 IP[172.16.2.144] GNB_ID[0xc] (.../src/amf/ngap-handler.c:176)
> 2022-04-26 12:59:36 NGSetupRequest (.../src/amf/ngap-handler.c:128)
> 2022-04-26 12:59:36 TO - - 172.16.2.133:38412 NG setup request
> 2022-04-26 12:59:36 - Connected to 172.16.2.133:38412
> 2022-04-26 12:59:35 - Connecting to 172.16.2.133:38412 | AMF
NGAP
```

Fig 35. Screenshot of NGAP workflows.

This screenshot shows the logs interleaved between Amarisoft and Open5Gs as the processes explained above are followed with the NGAP protocol.

4.2.1.2. UE added workflows

```

|> 2022-05-02 11:17:34 DL 0004 5GMM: Configuration update command
|> 2022-05-02 11:17:34 FROM 0005 0004 172.16.2.133:38412 Downlink NAS transport
|> 2022-05-02 11:17:34 IP[172.16.2.144] RAN_ID[12] (./src/amf/ngap-path.c:64)
|> 2022-05-02 11:17:34 RAN_UE_NGAP_ID[4] AMF_UE_NGAP_ID[5] (./src/amf/ngap-build.c:364)
|> 2022-05-02 11:17:34 DownlinkNASTransport (./src/amf/ngap-build.c:320)
|> 2022-05-02 11:17:34 [imsi-00103543200003] Configuration update command (./src/amf/nas-path.c:389)
|> 2022-05-02 11:17:34 Timezone:0x80 (./src/amf/gmm-build.c:521)
|> 2022-05-02 11:17:34 LOCAL [2022-05-02T11:17:34] Timezone[7200]/DST[1] (./src/amf/gmm-build.c:511)
|> 2022-05-02 11:17:34 UTC [2022-05-02T09:17:34] Timezone[0]/DST[0] (./src/amf/gmm-build.c:506)
|> 2022-05-02 11:17:34 RAN_UE_NGAP_ID[4] AMF_UE_NGAP_ID[5] TAC[67] CellID[0xcde] (./src/amf/ngap-handler.c:644)
|> 2022-05-02 11:17:34 [imsi-00103543200003] Registration complete (./src/amf/gmm-sm.c:988)
|> 2022-05-02 11:17:34 IP[172.16.2.144] RAN_ID[12] (./src/amf/ngap-handler.c:563)
|> 2022-05-02 11:17:34 UplinkNASTransport (./src/amf/ngap-handler.c:540)
|> 2022-05-02 11:17:34 RAN_UE_NGAP_ID[4] AMF_UE_NGAP_ID[5] (./src/amf/ngap-handler.c:837)
|> 2022-05-02 11:17:34 IP[172.16.2.144] RAN_ID[12] (./src/amf/ngap-handler.c:805)
|> 2022-05-02 11:17:34 InitialContextSetupResponse (./src/amf/ngap-handler.c:784)

|> 2022-05-02 11:17:34 UL 0004 5GMM: Registration complete
|> 2022-05-02 11:17:34 TO 0005 0004 172.16.2.133:38412 Uplink NAS transport
|> 2022-05-02 11:17:34 TO 0005 0004 172.16.2.133:38412 Initial context setup response
|> 2022-05-02 11:17:34 DL 0004 5GMM: Registration accept
|> 2022-05-02 11:17:34 FROM 0005 0004 172.16.2.133:38412 Initial context setup request
|> 2022-05-02 11:17:34 IP[172.16.2.144] RAN_ID[12] (./src/amf/ngap-path.c:64)
|> 2022-05-02 11:17:34 RAN_UE_NGAP_ID[4] AMF_UE_NGAP_ID[5] (./src/amf/ngap-build.c:625)
|> 2022-05-02 11:17:34 InitialContextSetupRequest(UE) (./src/amf/ngap-build.c:475)

|> 2022-05-02 11:17:34 [imsi-00103543200003] Registration accept (./src/amf/nas-path.c:76)
|> 2022-05-02 11:17:34 [imsi-00103543200003] SERVED_TAI_INDEX[0] (./src/amf/gmm-build.c:96)
|> 2022-05-02 11:17:34 [imsi-00103543200003] NR_CGI[PLMN_ID:00f130,CELL_ID:0xcde] (./src/amf/gmm-build.c:93)
|> 2022-05-02 11:17:34 [imsi-00103543200003] TAI[PLMN_ID:00f130,TAC:67] (./src/amf/gmm-build.c:90)
|> 2022-05-02 11:17:34 [imsi-00103543200003] 5G-S_GUTI[AMF_ID:0x20040,M_TMSI:0xc0001cde] (./src/amf/gmm-build.c:77)
|> 2022-05-02 11:17:34 5GMM Capability:[LPP:0, HO_ATTACH:0, S1_MODE:0] (./src/amf/gmm-handler.c:314)
|> 2022-05-02 11:17:34 Registration request in NAS message container (./src/amf/gmm-handler.c:1134)
|> 2022-05-02 11:17:34 SERVED_TAI_INDEX[0] (./src/amf/gmm-handler.c:246)
|> 2022-05-02 11:17:34 NR_CGI[PLMN_ID:00f130,CELL_ID:0xcde] (./src/amf/gmm-handler.c:227)
|> 2022-05-02 11:17:34 TAI[PLMN_ID:00f130,TAC:67] (./src/amf/gmm-handler.c:224)
|> 2022-05-02 11:17:34 OLD_NR_CGI[PLMN_ID:00f130,CELL_ID:0xcde] (./src/amf/gmm-handler.c:221)
|> 2022-05-02 11:17:34 OLD TAI[PLMN_ID:00f130,TAC:67] (./src/amf/gmm-handler.c:218)
|> 2022-05-02 11:17:34 NEW TSC[UE:0,AMF:0] KSI[UE:1,AMF:1] (./src/amf/gmm-handler.c:195)
|> 2022-05-02 11:17:34 OLD TSC[UE:0,AMF:0] KSI[UE:1,AMF:1] (./src/amf/gmm-handler.c:188)
|> 2022-05-02 11:17:34 [suci-0-001-03-0-0-0-543200003] 5G-S_GUTI[AMF_ID:0x20040,M_TMSI:0xe100e14f] (./src/amf/gmm-handler.c:171)
|> 2022-05-02 11:17:34 [suci-0-001-03-0-0-0-543200003] Known UE by 5G-S_TMSI[AMF_ID:0x20040,M_TMSI:0xe100e14f] (./src/amf/context.c:135)
|> 2022-05-02 11:17:34 Registration request (./src/amf/gmm-sm.c:130)
|> 2022-05-02 11:17:34 RAN_UE_NGAP_ID[4] AMF_UE_NGAP_ID[5] TAC[67] CellID[0xcde] (./src/amf/ngap-handler.c:500)
|> 2022-05-02 11:17:34 IP[172.16.2.144] RAN_ID[12] (./src/amf/ngap-handler.c:388)
|> 2022-05-02 11:17:34 [Added] Number of gNB-UEs is now 1 (./src/amf/context.c:1961)
|> 2022-05-02 11:17:34 InitialUEMessage (./src/amf/ngap-handler.c:361)

|> 2022-05-02 11:17:34 UL 0004 5GMM: Registration request
|> 2022-05-02 11:17:34 TO - 0004 172.16.2.133:38412 Initial UE message

```

Fig 36. Screenshot of the workflow with UE.

In this case, you can see all the processes of both protocols, both NGAP and NAS to add a UE. Fig 36 shows how they follow the same processes explained previously. The Quectel modem is used as a UE to connect to a new APN and obtain an IP using the commands defined in the following table.

Command	Function
qmcli -p -d /dev/cdc-wdm0 --wds-start-network="apn=internet" --client-no-release-cid	Initiating connection via APN.
udhcpc -i wwan0	Obtaining IP.

Table 4. Commands used in Quectel modem to create connection.

4.2.2. my5G-RANTester experiments

It is necessary to perform tests with all NFs, trying a high number of combinations. This way we have more knowledge of the network and in case of a future failure during deployment, we will have a programming based solution to automate the solution of that failure. Each test was run three times to check for variations in the workflows. Therefore, before each test the following is performed:

- Stop my5G-RANTester.
- Restart Open5Gs with the needed configs/status.
- Start my5G-RANTester with the needed configs/status.
- Capture all the logs (my5G-RANTester and all the NFs of Open5Gs).
- Note the time of each test (for finding it later in Grafana).
- Logs in Open5Gs at debug level with the necessary filters.

We decided to perform these tests with the my5G-RANTester program because it is the most suitable software for all types of testing.

4.2.2.1. NG Setup without UE

In this case, the tests are performed with the gNB without the user, so we observe the registers that appear during the connection of the gNB and Open5Gs, i.e. during the NG Setup procedure.

Test number	Type of test	Result
1.1	PLMNID (00101), SNSSAI (SST 1, SD 000001) and TAC (1) are correct in all sides (gNB, AMF, NSSF)	All correct
1.2	Wrong PLMNID in gNB	AMF indicates TAI error
1.3	Wrong SST-SD in gNB	AMF indicates NSSAI error
1.4	Wrong TAC in gNB	AMF indicates TAI error
1.5	Wrong SST-SD in NSSF	No error logs appear

Table 5. Summary of NG Setup tests without UE.

It is decided to analyze the screenshots of the most relevant tests, in this case the images of test numbers 1.1, 1.2 and 1.3 will be shown below. All numbers specified in the screenshots refer to what was previously studied in chapter 2.1.6 of Fig 6.

a. Test 1.1 - All parameters correct

```
> 2022-06-15 11:17:56 [GNB] Control interface IP/Port: 127.0.0.1/9487"
> 2022-06-15 11:17:56 [GNB] Number of GNBS: 1"
> 2022-06-15 11:17:56 Starting test function: Testing an gnb attached with configuration"
> 2022-06-15 11:17:56     IP[127.0.0.1] RAN_ID[1] (.../src/amf/ngap-path.c:63)
> 2022-06-15 11:17:56 NGSetupResponse (.../src/amf/ngap-build.c:36) 2a
> 2022-06-15 11:17:56 NG-Setup response (.../src/amf/ngap-path.c:244)
> 2022-06-15 11:17:56     S_NSSAI[SST:1 SD:0xffffffff] (.../src/amf/ngap-handler.c:73)
> 2022-06-15 11:17:56     PLMN_ID[MCC:1 MNC:1] (.../src/amf/ngap-handler.c:68)
> 2022-06-15 11:17:56     SERVED_TAI_INDEX[0] (.../src/amf/ngap-handler.c:46)
> 2022-06-15 11:17:56     PLMN_ID[MCC:1 MNC:1] (.../src/amf/ngap-handler.c:41)
> 2022-06-15 11:17:56     TAC[1] (.../src/amf/ngap-handler.c:40)
> 2022-06-15 11:17:56     S_NSSAI[SST:1 SD:0xffffffff] (.../src/amf/ngap-handler.c:269)
> 2022-06-15 11:17:56     PLMN_ID[MCC:1 MNC:1] (.../src/amf/ngap-handler.c:230)
> 2022-06-15 11:17:56     TAC[1] (.../src/amf/ngap-handler.c:202)
> 2022-06-15 11:17:56     PagingDRX[2] (.../src/amf/ngap-handler.c:179)
> 2022-06-15 11:17:56     IP[127.0.0.1] GNB_ID[0x1] (.../src/amf/ngap-handler.c:176)
> 2022-06-15 11:17:56 NGSetupRequest (.../src/amf/ngap-handler.c:128) 1
> 2022-06-15 11:17:56 SCTP_COMM_UP (.../src/amf/ngap-sctp.c:161)
> 2022-06-15 11:17:56 SCTP_ASSOC_CHANGE:[T:32769, F:0x0, S:0, I/O:2/2] (.../src/amf/ngap-sctp.c:152)
> 2022-06-15 11:17:56 gNB-N2[127.0.0.1] max_num_of_ostreams : 2 (.../src/amf/amf-sm.c:658)
> 2022-06-15 11:17:56 [Added] Number of gNBs is now 1 (.../src/amf/context.c:876)
```

Fig 37. Screenshot of test 1.1

In this case all parameters are correct in the gNB with the PLMNID and in the AMF and NSSF NFs of the 5GCore they are also correct, as the mnc/mcc match and the Core supports Slice 1.

First, the amf sends an NGSetupRequest (1) message and then receives the NGSetupResponse (2) reply from the gNB to the AMF.

b. Test 1.2 - Wrong PLMNID in gNB

```
> 2022-06-15 11:53:17 NG-Setup failure (.../src/amf/ngap-path.c:260)
> 2022-06-15 11:53:17     S_NSSAI[SST:1 SD:0xffffffff] (.../src/amf/ngap-handler.c:269)
> 2022-06-15 11:53:17     PLMN_ID[MCC:3 MNC:3] (.../src/amf/ngap-handler.c:230)
> 2022-06-15 11:53:17     TAC[1] (.../src/amf/ngap-handler.c:202)
> 2022-06-15 11:53:17     PagingDRX[2] (.../src/amf/ngap-handler.c:179)
> 2022-06-15 11:53:17     IP[127.0.0.11] GNB_ID[0x1] (.../src/amf/ngap-handler.c:176)
> 2022-06-15 11:53:17 NGSetupRequest (.../src/amf/ngap-handler.c:128) 1
> 2022-06-15 11:53:17     Cannot find Served TAI. Check 'amf.tai' configuration (.../src/amf/ngap-h
> 2022-06-15 11:53:17 NG-Setup failure: (.../src/amf/ngap-handler.c:305) 2b
> 2022-06-15 11:53:17 SCTP_COMM_UP (.../src/amf/ngap-sctp.c:161)
> 2022-06-15 11:53:17 SCTP_ASSOC_CHANGE:[T:32769, F:0x0, S:0, I/O:2/2] (.../src/amf/ngap-sctp.c:152)
> 2022-06-15 11:53:17 gNB-N2[127.0.0.11] max_num_of_ostreams : 2 (.../src/amf/amf-sm.c:658)
> 2022-06-15 11:53:17 [Added] Number of gNBs is now 1 (.../src/amf/context.c:876)
```

Fig 38. Screenshot of test 1.2

Test 1.2 shows the NGAP protocol request message (1) and the failure response (2b). This workflow also agrees with what has been studied. The reason is that in the gNB part of my5G-RANTester the mnc/mcc parameters are wrong because they are not identical to those of Open5Gs.

c. Test 1.3 - Wrong SST-SD in gNB

```

> 2022-06-16 10:08:31 S_NSSAI[SST:3 SD:0xffffffff] (.../src/amf/ngap-handler.c:269)
> 2022-06-16 10:08:31 PLMN_ID[MCC:1 MNC:1] (.../src/amf/ngap-handler.c:230)
> 2022-06-16 10:08:31 TAC[1] (.../src/amf/ngap-handler.c:202)
> 2022-06-16 10:08:31 PagingDRX[2] (.../src/amf/ngap-handler.c:179)
> 2022-06-16 10:08:31 IP[127.0.0.10] GNB_ID[0x1] (.../src/amf/ngap-handler.c:176)
> 2022-06-16 10:08:31 NGSetupRequest (.../src/amf/ngap-handler.c:128) 1
> 2022-06-16 10:08:31 Cannot find S_NSSAI. Check 'amf.plmn_support.s_nssai' configuration
> 2022-06-16 10:08:31 NG-Setup failure: (.../src/amf/ngap-handler.c:316) 2b

```

Fig 39. Screenshot of test 1.3

It is decided to put a slice in the gNB that does not support Open5gs, since the Core configuration only supports `sst:1` and `sst:2`, but in this case `sst:3` is used as shown in Fig 39. Two errors are observed in Loki, in which the first log the AMF indicates that there is an error, however, the relevant log is the one that says the AMF itself indicating “*Cannot find S_NSSAI. Check ‘amf.plmn_support.s_nssai’ configurations*”.

d. Test 1.4 - Wrong TAC in gNB

The TAC is part of the TAI, therefore exactly the same error appears as if we fail in the mnc/mcc.

e. Test 1.5 - Wrong SST-SD in NSSF

No error is observed because it will not be detected until a UE tries to connect. Later, in test 3.3, if a UE tries to connect, the error will appear.

4.2.2.2. UE connection + UE credentials

This section analyses the workflows when a UE connects, varying its credentials.

Test number	Type of test	Result
2.1	PLMNID (00101), SNSSAI (SST 1, SD 000001) and TAC (1) are correct in all sides (gNB, AMF, NSSF)	All correct
2.2	Ki parameter wrong in UE or in Open5Gs DB	AMF indicates Authentication failure
2.3	OPc parameter wrong in UE or in Open5Gs DB	AMF indicates Authentication failure
2.4	APN parameter wrong in the UE or in Open5Gs DB	AMF indicates DNN error
2.5	SST-SD parameter wrong in the UE or in Open5Gs DB	AMF indicates DNN error

Table 6. Summary of UE connection with credentials tests.

It is decided to analyze the captures of tests 2.1, 2.2 and 2.4, as the other tests have repeated logs. NGAP messages are ignored, as they have been analyzed in the previous test, so we focus on NAS messages to analyze the workflow of both protocols. All numbers specified in the screenshots refer to what was previously studied in chapter 2.1.6 of Fig 5.

a. Test 2.1 - All parameters correct

```

> 2022-06-16 11:51:42 DownlinkNASTransport (.../src/amf/ngap-build.c:320)
> 2022-06-16 11:51:42 [suci-0-001-01-0-0-0-0-0-0000000002] Authentication request 2 (.../src/amf/nas-path.c:314)
> 2022-06-16 11:51:42 [ddcc942-ed59-41ec-82e0-576381d6d1df] (NF-discover) NF Profile updated (.../src/udm
> 2022-06-16 11:51:42 [ddcc942-ed59-41ec-82e0-576381d6d1df] (NF-discover) NF registered (.../src/udm/nrnf
> 2022-06-16 11:51:42 [ddcc942-ed59-41ec-82e0-576381d6d1df:0] NF-Discovered [NF-Type:UDR,NF-Status:REGIS
> 2022-06-16 11:51:42 NF-Discover : Requester[UDM] Target[UDR] (.../src/nrf/nrnf-handler.c:425)
> 2022-06-16 11:51:42 Try to discover [UDR] (.../lib/sbi/path.c:110)
> 2022-06-16 11:51:42 [ddc18244-ed59-41ec-97f1-21e02236aaeb] (NF-discover) NF Profile updated (.../src/ausf
> 2022-06-16 11:51:42 [ddc18244-ed59-41ec-97f1-21e02236aaeb] (NF-discover) NF registered (.../src/ausf/nrnf
> 2022-06-16 11:51:42 [ddc18244-ed59-41ec-97f1-21e02236aaeb:0] NF-Discovered [NF-Type:UDM,NF-Status:REGIS
> 2022-06-16 11:51:42 NF-Discover : Requester[AUSF] Target[UDM] (.../src/nrf/nrnf-handler.c:425)
> 2022-06-16 11:51:42 [Added] Number of AMF-UEs is now 1 (.../src/amf/context.c:1197)
> 2022-06-16 11:51:42 [suci-0-001-01-0-0-0-0-0000000002] Unknown UE by SUCI (.../src/amf/context.c:1398)
> 2022-06-16 11:51:42 Try to discover [UDM] (.../lib/sbi/path.c:110)
> 2022-06-16 11:51:42 [suci-0-001-01-0-0-0-0-0000000002] SUCI (.../src/amf/gmm-handler.c:157)
> 2022-06-16 11:51:42 Registration request (.../src/amf/gmm-sm.c:134) 1
> 2022-06-16 11:51:42 IP[127.0.0.1] RAN_ID[1] (.../src/amf/ngap-handler.c:387)

```

Fig 40. Screenshot of test 2.1 part 1

The AMF receives the registration request record from the gNB (1), and then sends the authentication request record (2) to verify that the user is authenticated to complete the registration.

```

> 2022-06-16 11:51:42 UplinkNASTransport (.../src/amf/ngap-handler.c:540) 8
> 2022-06-16 11:51:42 [imsi-0010100000000002] Configuration update command (.../src/amf/nas-path.c:389)
> 2022-06-16 11:51:42 LOCAL [2022-06-16T11:51:42] Timezone[7200]/DST[1] (.../src/amf/gmm-build.c:507)
> 2022-06-16 11:51:42 UTC [2022-06-16T09:51:42] Timezone[0]/DST[0] (.../src/amf/gmm-build.c:502)
> 2022-06-16 11:51:42 [imsi-0010100000000002] Registration complete (.../src/amf/gmm-sm.c:1063) 7
> 2022-06-16 11:51:42 [NGAP][AMF] Send Initial Context Setup Response. 5a

```

Fig 41. Screenshot of test 2.1 part 2

Once the user has been authenticated, registration (7) is performed and the commands (8) are configured to finally establish the connection.

```

> 2022-06-16 11:51:42 Session Modification Request (.../src/smf/n4-build.c:132)
> 2022-06-16 11:51:42 PDUSESSIONResourceSetupResponseTransfer 10 (.../src/smf/ngap-handler.c:52)
> 2022-06-16 11:51:42 RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[1] (.../src/amf/ngap-handler.c:1565)
> 2022-06-16 11:51:42 IP[127.0.0.1] RAN_ID[1] (.../src/amf/ngap-handler.c:1533)
> 2022-06-16 11:51:42 PDUSESSIONResourceSetupResponse (.../src/amf/ngap-handler.c:1508)
> 2022-06-16 11:51:42 IP[127.0.0.1] RAN_ID[1] (.../src/amf/ngap-path.c:63)
> 2022-06-16 11:51:42 RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[1] (.../src/amf/ngap-build.c:1370)
> 2022-06-16 11:51:42 PDUSESSIONResourceSetupRequest(Session) 9 (.../src/amf/ngap-build.c:1325)
> 2022-06-16 11:51:42 [ddcb5698-ed59-41ec-bfac-2320649442b8] (NF-discover) NF Profile updated (

```

Fig 42. Screenshot of test 2.1 part 3

Finally the AMF makes a PDU Request (9) and the AMF responds that it accepts the PDU Session Stably Accept (10).

b. Test 2.2 - Ki parameter wrong

```
> 2022-06-17 09:53:34      RAN UE NGAP_ID[1] AMF UE NGAP_ID[1] (.../src/amf/ngap-path.c:313)
> 2022-06-17 09:53:34  UEContextReleaseCommand (.../src/amf/ngap-path.c:312)
> 2022-06-17 09:53:34      IP[127.0.0.1] RAN_ID[1] (.../src/amf/ngap-path.c:63)
> 2022-06-17 09:53:34      RAN UE NGAP_ID[1] AMF UE NGAP_ID[1] (.../src/amf/ngap-build.c:363)
> 2022-06-17 09:53:34 DownlinkNASTransport (.../src/amf/ngap-build.c:320)
> 2022-06-17 09:53:34      RAN UE NGAP_ID[1] AMF UE NGAP_ID[1] TAC[1] CellID[0x1] (.../src/amf/ngap-path.c:313)
> 2022-06-17 09:53:34      IP[127.0.0.1] RAN_ID[1] (.../src/amf/ngap-handler.c:562)
> 2022-06-17 09:53:34 UplinkNASTransport (.../src/amf/ngap-handler.c:540) 4b
> 2022-06-17 09:53:34 [suci-0-001-01-0-0-0-0000000002] Authentication reject (.../src/amf/nas-path.c:313)
> 2022-06-17 09:53:34 Authentication failure(MAC failure) 3c (.../src/amf/gmm-sm.c:531)
> 2022-06-17 09:53:34      IP[127.0.0.1] RAN_ID[1] (.../src/amf/ngap-path.c:63)
> 2022-06-17 09:53:34      RAN UE NGAP_ID[1] AMF UE NGAP_ID[1] (.../src/amf/ngap-build.c:363)
> 2022-06-17 09:53:34 DownlinkNASTransport (.../src/amf/ngap-build.c:320) 2c
> 2022-06-17 09:53:34 [suci-0-001-01-0-0-0-0000000002] Authentication request (.../src/amf/nas-path.c:313)
```

Fig 43. Screenshot of test 2.2

In this case an error is made in the Subscriber Key parameter, which is part of the authentication. The AMF sends the Authentication Request message to the gNB (2c) and replies to the AMF indicating that there is an Authentication failure error (3c), so it will notify this error to the AMF with the Authentication reject message (4b).

c. Test 2.4 - APN parameter wrong

```
> 2022-06-17 10:05:26 [imsi-0010100000000002] 5GMM status (.../src/amf/nas-path.c:487)
> 2022-06-17 10:05:26      RAN UE NGAP_ID[1] AMF UE NGAP_ID[1] TAC[1] CellID[0x1] (.../src/amf/ngap-path.c:313)
> 2022-06-17 10:05:26      IP[127.0.0.10] RAN_ID[1] (.../src/amf/ngap-handler.c:562)
> 2022-06-17 10:05:26 UplinkNASTransport (.../src/amf/ngap-handler.c:540)
> 2022-06-17 10:05:26 [Added] Number of AMF-Sessions is now 1 (.../src/amf/context.c:2084) 10b
> 2022-06-17 10:05:26 [imsi-0010100000000002] DNN Not Supporere OR Not Subscribed in the Slice
> 2022-06-17 10:05:26      IP[127.0.0.10] RAN_ID[1] (.../src/amf/ngap-path.c:63)
> 2022-06-17 10:05:26      RAN UE NGAP_ID[1] AMF UE NGAP_ID[1] (.../src/amf/ngap-build.c:363)
> 2022-06-17 10:05:26 DownlinkNASTransport (.../src/amf/ngap-build.c:320)
> 2022-06-17 10:05:26      RAN UE NGAP_ID[1] AMF UE NGAP_ID[1] TAC[1] CellID[0x1] (.../src/amf/ngap-path.c:313)
> 2022-06-17 10:05:26      IP[127.0.0.10] RAN_ID[1] (.../src/amf/ngap-handler.c:562)
> 2022-06-17 10:05:26 UplinkNASTransport (.../src/amf/ngap-handler.c:540) 9
> 2022-06-17 10:05:26 [imsi-0010100000000002] Configuration update command 8 (.../src/amf/nas-path.c:313)
> 2022-06-17 10:05:26      LOCAL [2022-06-17T10:05:26] Timezone[7200]/DST[1] (.../src/amf/gmm-build.c:1063)
> 2022-06-17 10:05:26      UTC [2022-06-17T08:05:26] Timezone[0]/DST[0] (.../src/amf/gmm-build.c:1063)
> 2022-06-17 10:05:26 [imsi-0010100000000002] Registration complete (7) (.../src/amf/gmm-sm.c:1063)
```

Fig 44. Screenshot of test 2.4

APN (Access Point Name) is the name of the access point to which the user wants to connect. We analyze the workflow from the Registration Complete message (7), message sent by the gNB and received by the AMF, then it will send the Configuration Update Command message (8) to check that it supports all the requirements requested by the user. The next step is to make the request to establish the PDU session and the Uplink NAS Transport (9), where finally the AMF will say that it is not possible by means of a log that shows the user's IMSI and an error message in the DNN (Data Network Name).

4.2.2.3. Conexión UE + Slicing

The aim of this section is to observe errors that appear in the core when the UE tries to access an undefined PLMNID or slice.

Test number	Type of test	Result
3.1	UE with a wrong PLMNID	AMF, UDR, UDM and AUSF indicate error
3.2	UE with a wrong SST-SD	AMF indicates DNN/Slice error
3.3	UE, AMF and gNB with the same config but NSFF with a different SST-SD config	Sometimes NSSF indicates the error

Table 7. Summary of UE connection with Slicing tests.

In this section we will analyze the NAS protocol workflows (Fig 5) and show the screenshots of tests 3.1 and 3.3, since they are the most relevant because 3.2 are similar to test 2.5.

a. Test 3.1 - UE with a wrong PLMNID

```

> 2022-06-17 14:21:26 [suci-0-003-03-0-0-0-0000000001] Registration reject [11] (./src/amf/nas-path.c:154) 2a
> 2022-06-17 14:21:26 [suci-0-003-03-0-0-0-0000000001] Cannot find SUCI [404] (./src/amf/gmm-sm.c:649)
> 2022-06-17 14:21:26 [suci-0-003-03-0-0-0-0000000001]     SUCI (./src/amf/gmm-handler.c:157)
> 2022-06-17 14:21:26 Registration request (./src/amf/gmm-sm.c:134) 1
> 2022-06-17 14:21:26 [Added] Number of AMF-UEs is now 1 (./src/amf/context.c:1197)
> 2022-06-17 14:21:26 [suci-0-003-03-0-0-0-0000000001] Unknown UE by SUCI (./src/amf/context.c:1398)
> 2022-06-17 14:21:26     RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[1] TAC[1] CellID[0x1] (./src/amf/ngap-handler.c:497)
> 2022-06-17 14:21:26 [Added] Number of gNB-UEs is now 1 (./src/amf/context.c:2072)
> 2022-06-17 14:21:26     IP[127.0.0.11] RAN_ID[1] (./src/amf/ngap-handler.c:387)
> 2022-06-17 14:21:26 InitialUEMessage (./src/amf/ngap-handler.c:361)
> 2022-06-17 14:21:26 [imsi-0-00303000000001] Cannot find SUPI in DB (./src/udr/nudr-handler.c:69) UDR
> 2022-06-17 14:21:26 [imsi-0-00303000000001] Cannot find IMSI in DB (./lib/dbi/subscription.c:56)
> 2022-06-17 14:21:26 [f70880a2-ee37-41ec-8083-670d06e355d9:0] NF-Discover : [NF-Type:UDR,NF-Status:REGISTERED,IPv4:1,IP]
> 2022-06-17 14:21:26 NF-Discover : Requester[UDM] Target[UDR] (./src/rnf/nnrf-handler.c:425)
> 2022-06-17 14:21:26 [suci-0-003-03-0-0-0-0000000001] HTTP response error [404] (./src/udm/nudr-handler.c:86) UDM
> 2022-06-17 14:21:26 [f70880a2-ee37-41ec-8083-670d06e355d9] (NF-discover) NF Profile updated (./src/udm/nnrf-handler.c:286)
> 2022-06-17 14:21:26 [f70880a2-ee37-41ec-8083-670d06e355d9] (NF-discover) NF registered (./src/udm/nnrf-handler.c:286)
> 2022-06-17 14:21:26 Try to discover [UDR] (./lib/sbi/path.c:110)
> 2022-06-17 14:21:26 [NGAP] Receive Downlink NAS Transport"
> 2022-06-17 14:21:26 [SCTP] Receive message in 1 stream\n"
> 2022-06-17 14:21:26 [SCTP] Receive message in 1 stream\n"
> 2022-06-17 14:21:26 [suci-0-003-03-0-0-0-0000000001] Cannot find SUPI [404] (./src/ausf/ue-sm.c:126) AUSF

```

Fig 45. Screenshot of test 3.1

In this test, errors appear in the AMF, UDR, UDM and AUSF components. However, only the AMF workflow will be analyzed, since it is part of the NAS protocol.

b. Test 3.3 - NSSF with a different SST

```
|> 2022-06-23 11:53:16 [imsi-00101000000001] HTTP response error [403] (./src/amf/nssf-handler.c:43)
|> 2022-06-23 11:53:16 Cannot find NSI by S-NSSAI[SST:1 SD:0xffffffff] (./src/nssf/nssf-handler.c:89)
|> 2022-06-23 11:53:16 [401fcbb46-f2da-41ec-977d-6f5d591f10c7] (NF-discover) NF Profile updated (./src/amf/nssf/nssf-discover.c:100)
```

Fig 46. Screenshot of test 3.3

This test consists of the UE, AMF and gNB having the same SST configuration, but the NSSF having a different configuration. After performing the same test three times, it is observed that there are times when the NSSF component detects the error and there are other times when no error is detected. This is probably an internal error in the code of Open5Gs.

4.2.2.4. UE connection + Session Creation

This test consists of configuring the SMF information, adding the S-NSSAI parameters (sst and ssd) and the DNN. The NAS protocol messages shall be analyzed.

Test number	Type of test	Result
4.1	SNSSAI (SST 1, SD 000001) in SMF	All correct
4.2	SNSSAI field in SMF info is wrong	AMF indicates that cannot discover SMF
4.3	DNN field in SMF info is wrong	AMF indicates that cannot discover SMF

Table 8. Summary of UE connection with session creation tests.

In this case, it is only interesting to analyze test 4.2, as the other tests are similar to the tests done previously.

a. Test 4.2

```
|> 2022-06-23 10:42:45 DownlinkNASTransport (./src/amf/ngap-build.c:320)
|> 2022-06-23 10:42:45 NF-Discover failed [400] (./src/amf/sbi-path.c:229)
|> 2022-06-23 10:42:45 [suci-0-001-01-0-0-0-0-0000000001] DL NAS transport (./src/amf/nas-path.c:573) 10b
|> 2022-06-23 10:42:45 Invalid API name [nnrf-disc] (./src/nssf/nssf-sm.c:130)
|> 2022-06-23 10:42:45     RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[1] TAC[1] CellID[0x1] (./src/amf/ngap-handler.c:100)
```

Fig 47. Screenshot of test 4.2

The error occurs when the AMF sends a message to the PDU Session Establishment Request SMF to check that the SMF supports the UE request. As this is not the case, the SMF sends to the AMF a DL NAS Transport + PDU Session Establishment Reject (10b) message as discussed in chapter 2.1.6, that's why in test 4.3 the same error appears.

4.2.2.5. UE connection + NFs stopped

In this section, the NF is first stopped and then the UE is connected to check errors during this process. The idea is to observe how a failure is detected when an NF is disconnected or when it is already deployed and there is a communication error between them. As it is a totally distributed system, it is possible that the NFs are in different machines, some components in the cloud, others in the edge, etc... The following table will show all tests

Test number	Type of test	Result
5.1	NSSF stopped before UE connection.	No logs appear, all correct.
5.2	SMF stopped before UE connection.	NSSF indicates the S-NSSAI error and AMF indicates HTTP error.
5.3	UPF stopped before UE connection.	SMF indicates that no UPFs are PFCP associated.
5.4	AUSF stopped before UE connection.	AMF indicates AUSF error.
5.5	UDR stopped before UE connection.	UDM indicates UDR error.
5.6	UDM stopped before UE connection.	AUSF indicates UDM error.
5.7	PCF stopped before UE connection.	AMF indicates PCF error.
5.8	SMF stopped during UE connection.	UPF indicates NO HEARTBEAT from SMF.
5.9	UPF stopped during UE connection.	SMF indicates NO HEARTBEAT from UPF.

Table 9. Summary of UE connection with NFs stopped tests.

In all cases an error is detected during the connection process. Only for the NSSF (5.2) case no error was detected, this is because it is not a mandatory NF and the 5G Core can provide basic functionality without that component.

There are two tests (5.8 and 5.9) where the NF is stopped during UE communication. In both cases errors are detected and are problematic. Two cases are shown in detail below:

a. Test 5.4

```
> 2022-06-21 09:34:40      IP[127.0.0.15] RAN_ID[1] (./src/amf/ngap-path.c:63)
> 2022-06-21 09:34:40      RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[1] (./src/amf/ngap-build.c:363)
> 2022-06-21 09:34:40 DownlinkNASTransport (./src/amf/ngap-build.c:320)
> 2022-06-21 09:34:40 [suci-0-001-01-0-0-0-0-000000001] (NF discover) No [AUSF] (./src/amf/nnrf-handler.c:285)
> 2022-06-21 09:34:40 [suci-0-001-01-0-0-0-0-000000001] Registration reject [90] (./src/amf/nas-path.c:154) 2a
> 2022-06-21 09:34:40 NF-Discover : Requester[AMF] Target[AUSF] (./src/nrf/nnrf-handler.c:425)
> 2022-06-21 09:34:40      IP[127.0.0.15] RAN_ID[1] (./src/amf/ngap-handler.c:387)
> 2022-06-21 09:34:40 [Added] Number of AMF-UEs is now 1 (./src/amf/context.c:1197)
> 2022-06-21 09:34:40 [suci-0-001-01-0-0-0-0-000000001] Unknown UE by SUCI (./src/amf/context.c:1398)
> 2022-06-21 09:34:40      RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[1] TAC[1] CellID[0x1] (./src/amf/ngap-handler.c:497)
> 2022-06-21 09:34:40 [Added] Number of gNB-UEs is now 1 (./src/amf/context.c:2072)
> 2022-06-21 09:34:40 InitialUEMessage (./src/amf/ngap-handler.c:361)
```

Fig 48. Screenshot of test 5.4

Initially the connection among NFs had been established correctly, however the AUSF component in Open5Gs stopped. The screenshot shows that, first, the AMF component asks the NRF for an AUSF, the NRF detects that it does not exist and notifies it, then the AMF throwssthe Registration reject (2a) error from the NAS protocol workflow. Finally the AMF responds to the NRF by saying No [AUSF].

b. Test 5.9

```
> 2022-06-21 10:11:46 [12] LOCAL No Reponse. Give up! for step 1 type 1 peer [127.0.0.7]:8805
> 2022-06-21 10:11:46 PFCP de-associated (./src/smf/pfcpc-sm.c:179)
> 2022-06-21 10:11:46 No Heartbeat from UPF [127.0.0.7]:8805 (./src/smf/pfcpc-sm.c:287)
> 2022-06-21 10:11:30 UPF terminate...done (./src/upf/app.c:39)
> 2022-06-21 10:11:30 [Removed] Number of UPF-sessions is now 0 (./src/upf/context.c:211)
```

Fig 49. Screenshot of test 5.9

This case is similar to the previous one, as the UPF component is stopped during the UE connection. There is no NAS or NGAP protocol error, first there is a UPF log announcing that the session has been dropped and then a SMF log announcing No Heartbeat from UPF. This refers to what was previously studied in chapter 2.1.2 where it is said that the SMF and the UPF communicate with each other.

4.2.2.6. Several Slices

In this section we look for slicing errors, analyzing different combinations of misconfigurations.

Test number	Type of test	Result
6.1	Connect UE1 (sst 1). Stop. Connect UE2 (sst 2).	No logs appear, all correct.
6.2	AMF only has one slice (e.g. sst 1).	AMF indicates NG-Setup failure.
6.3	NSSF only has one slice (e.g. sst 1).	Sometimes NSSF and AMF indicate error S_NSSAI.
6.4	SMF only has one slice.	UE1 (sst1) OK, UE2(sst2) KO AMF indicates SMF error.
6.5	Change configuration NSSF (sst:2) and restart only NSSF.	UE1 (SST:1) OK, UE2 (SST:2) ERROR, change configuration of NSSF (SST:2), restart only NSSF, connect UE2 (SST:2) OK.

Table 10. Summary of several slices tests.

The table shows that all tests show an error except for test 6.1, as both gNB and Core support the necessary slices for UE1 and UE2. It is interesting to analyze the behavior of the NSSF.

First, it does not always fail (6.3), which indicates a possible misimplementation as seen in another test before (3.3). The second interesting point is that in test 6.5 we see a case where an NF can be restarted (being a stateless NF), which opens the possibility of automation cases (e.g. we detect that the config is wrong, it is updated and restarted). Two cases are shown below:

a. Test 6.2

```
> 2022-06-28 16:15:30 NG-Setup failure (.../src/amf/ngap-path.c:260)
> 2022-06-28 16:15:30     Cannot find S_NSSAI. Check 'amf.plmn_support.s_nssai' configuration
> 2022-06-28 16:15:30 NG-Setup failure: (.../src/amf/ngap-handler.c:316) 2b
> 2022-06-28 16:15:30     SERVED_TAI_INDEX[0] (.../src/amf/ngap-handler.c:46)
> 2022-06-28 16:15:30     PLMN_ID[MCC:1 MNC:1] (.../src/amf/ngap-handler.c:41)
> 2022-06-28 16:15:30     TAC[1] (.../src/amf/ngap-handler.c:48)
> 2022-06-28 16:15:30     S_NSSAI[SST:2 SD:0xffffffff] (.../src/amf/ngap-handler.c:269)
> 2022-06-28 16:15:30     PLMN_ID[MCC:1 MNC:1] (.../src/amf/ngap-handler.c:230)
> 2022-06-28 16:15:30     TAC[1] (.../src/amf/ngap-handler.c:202)
> 2022-06-28 16:15:30     PagingDRX[2] (.../src/amf/ngap-handler.c:179)
> 2022-06-28 16:15:30     IP[127.0.0.1] GNB_ID[0x1] (.../src/amf/ngap-handler.c:176)
> 2022-06-28 16:15:30 NGSetupRequest (.../src/amf/ngap-handler.c:128)
```

Fig 50. Screenshot of test 6.2

In this case it is an NGAP protocol error, that the NGSetup can no longer be done because AMF and gNB have different slices. This is the same case as in test 1.3. The screenshot shows the first NGSetup Request message and then the NGSetup failure (2b). The AMF indicates the plmn_support error.

b. Test 6.4

```
> 2022-06-28 16:45:42 [suci-0-001-01-0-0-0-0000000002] DL NAS transport 10b/src/amf/nas-path.c:248
> 2022-06-28 16:45:42 Cannot discover [SMF] (./src/amf/sbi-path.c:248)
> 2022-06-28 16:45:42 [a9d8d9dc-f6f0-41ec-be18-3f02e3d2c802] (NF-discover) NF Profile updated
> 2022-06-28 16:45:42 NF EndPoint updated [127.0.0.4:7777] (./lib/sbi/context.c:1082)
```

Fig 51. Screenshot of test 6.4

This is a NAS protocol error, because the SMF component is the only one that contains only one slice (SST 1). Therefore, the NRF sends a NF Profile update log and the AMF replies with a Cannot discover [SMF] for the second slice (sst 2, UE 2). Finally the NAS error (10b) appears.

4.2.3. Experiment: automatic registration of a new user

This section demonstrates how this project automatically adds a user that is not registered in the Open5Gs database by using Grafana logs and alerts and by implementing an API which interacts with Open5Gs database, as was described in Section 4.1.4.

First, I check that the Open5Gs database is empty. In case a user tries to connect, errors will appear.

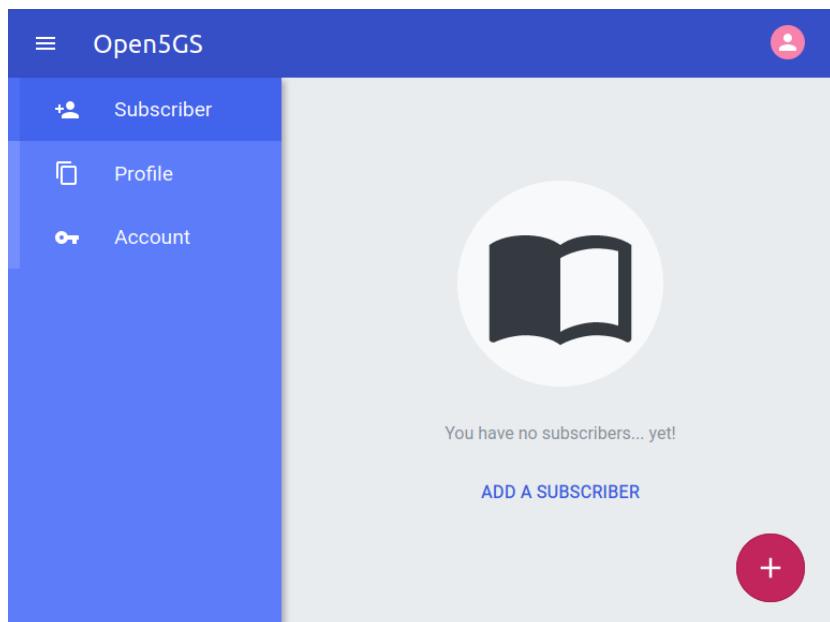


Fig 52. Empty Open5Gs MongoDB.

In the following image, it is also shown that our DB is also empty via mongoDB shell commands.

```

> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
newUE      0.000GB
open5gs    0.000GB
> use newUE
switched to db newUE
> show collections
imsi
> db.imsi.find()
>

```

Fig 53. My empty database.

The next step is to try to connect a user with the IMSI 001010000000001 to observe in Loki the error indicated by the UDR and the number of times the error appears, as shown in the following image.

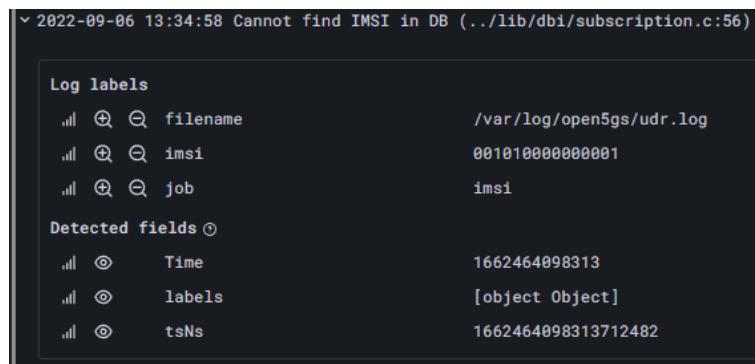


Fig 54. User error observed in Loki.

As mentioned in chapter 4.1.4.2, this error will trigger an alarm and through a Grafana Alert Manager it will send a POST to our API that will be waiting for this alarm to automate the process. The following image shows the logs that our code prints.

```

New IMSI 001010000000001 added
MongoDB shell version v3.6.8
connecting to: mongodb://localhost/open5gs
Implicit session: session { "id" : UUID("e5a0f72d-2b56-4c39-ab60-fa9558d41315") }
MongoDB server version: 3.6.8
WriteResult({
    "nMatched" : 0,
    "nUpserted" : 1,
    "nModified" : 0,
    "_id" : ObjectId("6317307b8da6b3cc8ed8a289")
})

172.18.0.4 - - [06/Sep/2022 13:35:23] "POST /alerts HTTP/1.1" 200 -
Ignored 001010000000001 IMSI

172.18.0.4 - - [06/Sep/2022 13:35:25] "POST /alerts HTTP/1.1" 200 -

```

Fig 55. Logs from my API.

As mentioned in chapter 4.1.3.4, the API created will add the new user to the Open5Gs database by running a script, it will also add to our database to identify the newly added users. On the other hand, it is also shown that if less than one minute has passed, the API will ignore the alarm.

Finally it is demonstrated how the user is automatically added to the Open5Gs database, as shown in the following image.

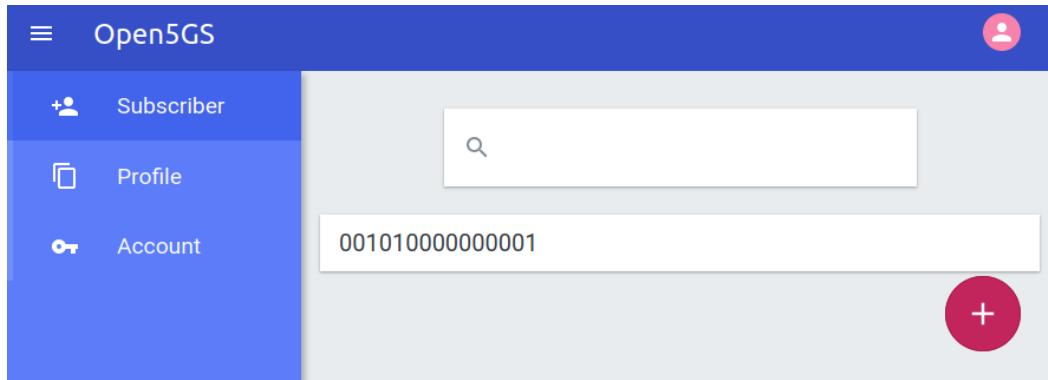


Fig 56. Updated Open5Gs database.

The following image shows how this new IMSI is added to my database with its respective timestamp defined in minutes.

```
> show collections
imsi
> db.imsi.find()
> db.imsi.find()
{ "_id" : ObjectId("63173bf77315949d644b5592"), "imsi" : "001010000000001", "date" : 27707784 }
```

Fig 57. My updated database.

The automation of adding a new user to the Open5Gs database is demonstrated using Loki alerts and sent to the API with AlertManager and Grafana.

CHAPTER 5

5.1. Conclusions

In this thesis we have studied the operation of the 5GCore, the components that form its new architecture, the workflows during its operation and the importance of the virtualisation of the disaggregated Core. This concept brings multiple benefits to the network, since if we have the UPF component close to the radio, the data will be transmitted very fast to the Internet. On the other hand, it also allows us to have the AMF and SMF components in charge of the control part, in a cloud away from the user. However, this adds complexity to deployments and, in environments such as private 5G, it is essential to minimize operational costs and avoid network downtime due to errors.

Observability and automation are proposed as a solution to the increasing complexity of the network. Therefore, this project focuses on monitoring through logs to analyze possible Core issues in different situations (e.g. UE attachment, slicing, NF misconfiguration and misoperations, etc...). With a good environment set up, tests have been successfully performed using RAN simulators (UERANSIM and my5G-RANTester), a 5GCore implementation (Open5GS) and even a real radio (Amarisoft). By means of multiple tests, errors were detected and a Zero-Touch Automation solution was demonstrated to add UEs to the Open5Gs database based on log detection, alert creation and API development.

This thesis has been carried out by using open source software (i.e. RAN simulators, Open5Gs, Grafana), so that anyone can replicate my configurations and tests. To the best of my knowledge, this is the first integration of Grafana Loki and Open5Gs to monitor logs and to create automations based on anomaly detections and corrections. Therefore, this work can be used as a basis for future research and innovation activities.

5.2. Future development

For future work, the experiment of adding a new user with a real RAN could be carried out, in order to analyze the operations of my API in a real environment. On the other hand, the analysis of all the problems mentioned in the project is basic and the existence of patterns have not been analyzed in detail because it would require many repetitions of the tests. Therefore, artificial intelligence could be integrated in the future to analyze patterns and automate more processes. The aim is to resolve log-based anomalies using a model based on deep learning, by conducting experiments that focus on several aspects of model evaluation, including training data selection, data grouping, class distribution, data noise, and early detection ability [4]. This type of AI-based automations will be key in beyond 5G and 6G networks.

Bibliography

- [1] Dimitris Giannopoulos, “Tutorial on communication between access networks and the 5G core”, Chapter 2.
- [2] Kleber Vieira Cardoso, Cristiano Bonato Both, Lucio Rene Prade, Ciro J. A. Macedo, Victor Hugo L. Lopes, “A softwarized perspective of the 5G networks”, Chapter 2.
- [3] Cindy Sridharan, “The pilars of Observability”, Chapter 4.
- [4] Van-Hoang Le, “Log Based anomaly detection”, Chapter 1.
- [5] Steffan Rommer, “5G core networks powering digitalization”, Chapter 3.
- [6] Grafana Labs, “Observability with logs”, <https://grafana.com/oss/loki/>
- [7] Open5Gs, “Quickstart”,
<https://open5gs.org/open5gs/docs/guide/01-quickstart/>

Annex

1. Amarisoft promtail file

```
---
```

```
server:
  http_listen_port: 9080
  http_listen_address: 0.0.0.0
  grpc_listen_port: 0

clients:
- url: http://84.88.36.127:3100/loki/api/v1/push
  tenant_id: tenant1

scrape_configs:
- job_name: amarisoft_ran
  static_configs:
    - targets:
      - localhost
      labels:
        job: amarisoft_ran
        __path__: /var/log/5g_sa.log

pipeline_stages:
- match:
    selector: '{job="amarisoft_ran"}'
    stages:
      - regex:
          expression: '(?P<timestamp>[0-9]{1,4}-[0-9]{1,2}-[0-9]{1,2}[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}.[0-9]{1,3}) \[(?P<component>.*)\] (?P<output>.*)'
          - labels:
              component:
            - output:
                source: output
            - timestamp:
                source: timestamp
                format: "2006-01-02 03:04:05.000"
                location: Europe/Madrid
```

2. Open5Gs promtail file

```
--
```

```
server:
  http_listen_port: 9080
  http_listen_address: 0.0.0.0
  grpc_listen_port: 0

clients:
- url: http://loki:3100/loki/api/v1/push
  tenant_id: tenant1

scrape_configs:
- job_name: open5gs
  static_configs:
```

```

- targets:
    - localhost
  labels:
    job: open5gs
    __path__: /var/log/open5gs/*.log

pipeline_stages:
- match:
    selector: '{job="open5gs"}'
    stages:
- regex:
        expression: '(?P<timestamp>[0-9]{1,2}\/[0-9]{1,2}
[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}.[0-9]{1,3}): \[(?P<component>.*?)\]
(?P<level>[A-Z]{1,8}): (?P<output>.*)'
        - labels:
            component:
            level:
- output:
    source: output
- timestamp:
    source: timestamp
    format: "01/02 03:04:05.000"
    location: Europe/Madrid

- drop:
    expression: '^ \(..'
- drop:
    expression: '^Open5GS'
- drop:
    expression: '.*_state_.*'
- drop:
    expression: 'PAA'
- drop:
    expression: '.*RECV.*'
- drop:
    expression: '.*QFI.*'

- match:
    selector: '{filename="/var/log/open5gs/amf.log"}'
    stages:
    - static_labels:
        nf: amf
- match:
    selector: '{filename="/var/log/open5gs/hss.log"}'
    stages:
    - static_labels:
        nf: hss

- match:
    selector: '{filename="/var/log/open5gs/udr.log"}'
    stages:
    - static_labels:

```

```

        nf: udr

- match:
    selector: '{filename="/var/log/open5gs/mme.log"}'
    stages:
    - static_labels:
        nf: mme

- match:
    selector: '{filename="/var/log/open5gs/sgwc.log"}'
    stages:
    - static_labels:
        nf: sgwc

- match:
    selector: '{filename="/var/log/open5gs/smf.log"}'
    stages:
    - static_labels:
        nf: smf

- match:
    selector: '{filename="/var/log/open5gs/pcrf.log"}'
    stages:
    - static_labels:
        nf: pcrf

- match:
    selector: '{filename="/var/log/open5gs/nssf.log"}'
    stages:
    - static_labels:
        nf: nssf

- match:
    selector: '{filename="/var/log/open5gs/nrf.log"}'
    stages:
    - static_labels:
        nf: nrf

- match:
    selector: '{filename="/var/log/open5gs/ausf.log"}'
    stages:
    - static_labels:
        nf: ausf

- match:
    selector: '{filename="/var/log/open5gs/upf.log"}'
    stages:
    - static_labels:
        nf: upf

- match:
    selector: '{filename="/var/log/open5gs/pcf.log"}'

```

```

stages:
- static_labels:
  nf: pcf

- match:
  selector: '{filename="/var/log/open5gs/bsf.log"}'
  stages:
- static_labels:
  nf: bsf

- match:
  selector: '{filename="/var/log/open5gs/udm.log"}'
  stages:
- static_labels:
  nf: udm

- match:
  selector: '{filename="/var/log/open5gs/sgwu.log"}'
  stages:
- static_labels:
  nf: sgwu

```

3. my5G-RANTester promtail file

```

server:
  http_listen_port: 9080
  http_listen_address: 0.0.0.0
  grpc_listen_port: 0

clients:
- url: http://loki:3100/loki/api/v1/push
  tenant_id: tenant1

scrape_configs:
- job_name: my5G_RAN_tester
  static_configs:
    - targets:
      - localhost
    labels:
      job: my5G_RAN_tester
      __path__: /var/log/my5G_RAN_tester/logs_my5G_RAN_tester.txt

pipeline_stages:
- match:
  selector: '{job="my5G_RAN_tester"}'
  stages:
- regex:
    expression:
' [a-z]{1,4}=[0-9]{4}-[0-9]{2}-[0-9-A-Z]{5}:[0-9]{2}:[0-9]{2}\+[0-9]{2}:
[0-9]{2}" [a-z]{1,9}=(?P<level>[a-z]{1,9})
[a-z]{3}="\[(?P<component>.*?)\] (?P<output>.*)' 

```

```

- labels:
    component:
    level:
- output:
    source: output

- drop:
    source: "component"
    value: "UE"
- drop:
    expression: '----'
- drop:
    expression: 'my5G-RANTester version 0.1'
- drop:
    expression: 'UE]'

```

4. API code

```

from urllib import response
from flask import Flask, request, jsonify
import os
from pymongo import MongoClient
import datetime as DT

MONGO_URI = 'mongodb://localhost'
client = MongoClient(MONGO_URI)
db = client['newUE']
collection = db['imsi']

def minutes(timestamp=None):
    if timestamp is None:
        now = DT.datetime.now()
        timestamp = now.timestamp()
    return timestamp//60

app = Flask(__name__)

@app.route('/alerts', methods=['POST'])
def main():
    min = minutes()
    payload = request.get_json(force=True)
    alerts = []
    alerts = payload['alerts']
    for x in alerts:
        annotations=[]
        annotations = x['annotations']
        try:
            summary = []
            summary = annotations['summary']
            valor = summary.split("imsi=")[2]
            imsi = valor.split(',') [0]

```

```

        find = collection.find()
        results = list(find)
        imsis = dict()
        for y in collection.find():
            imsis[y['imsi']] = y['date']
        if (imsi not in imsis.keys()):
            print ("New IMSI "+imsi+" added")
            collection.insert_one({"imsi":imsi, "date" : min})
            os.system('bash ./script.sh add '+imsi+' '
465B5CE8B199B49FAA5F0A2EE238A6BC E8ED289DEBA952E4283B54E88E6183CA')
            exit()
        elif (imsis[imsi] + 1 < min):
            collection.delete_one({"imsi":imsi, "date" :
imsis[imsi] })
            print("IMSI "+imsi+" repeated but added to DB")
            collection.insert_one({"imsi":imsi, "date" : min})
            os.system('bash ./script.sh add '+imsi+' '
465B5CE8B199B49FAA5F0A2EE238A6BC E8ED289DEBA952E4283B54E88E6183CA')
            exit()
        else:
            print("Ignored "+imsi+" IMSI")
    except:
        print("")
    return 'success', 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)

```

