# Simulated 5G Core Network

## 1. Introduction and Overview

### 1.1 Purpose and Objectives

The primary objective of this project is to build an open-source, software-based comprehensive mobile core network implementation to serve as a platform for **5G research and experimentation**. This testbed aims to explore and validate the capabilities, features, and potential of 5G technology in a **controlled environment**. By integrating **Free5GC** (5G core) and **UERANSIM** (RAN and UE simulation), we enable a **diverse range of 5G scenarios**, including mobility management, handovers, and network analytics.

### 1.2 Key Components

* **Free5GC**: Provides essential control-plane and data-plane functions, ensuring compliance with **3GPP** standards while offering flexibility for customization.
* **UERANSIM**: Simulates the RAN (gNB) and UEs, allowing realistic testing of features like **mobility**, **handover**, and **network analytics**.

### 1.3 Target Audience

* **Developers and Researchers** interested in testing, extending, or analyzing 5G network functionalities.
* Academics exploring **mobility models**, **handover algorithms**, or **network data analytics** in 5G.

## 2. System Requirements

* **Hardware**: Computer.
* **Operating System**: Ubuntu 20.04(recommended) or above. With kernel preferably 5.0.0-23-generic or 5.4.x
* **Software:** Free5gc, UERANSIM
* **Programming Language**: Go (v1.21.8), Python, C++.

## 3. Testbed Architecture

This section details how the 5G core, RAN, and UE components interact.

<https://github.com/aligungr/UERANSIM/wiki>

### 3.1 Core Network (Free5GC)

The Free5GC core network is **modular** and **service-based** (SBA). Its **Network Repository Function (NRF)** serves as a registry for all Network Functions (NFs), which register and advertise their services via RESTful APIs.

* **AMF (Access and Mobility Management Function)**: Handles UE signaling (registration, mobility, handover management). Communicates with gNB via N2 interface.
* **SMF (Session Management Function)**: Manages PDU sessions (IP allocation, QoS policies), interacting with the **UPF (User Plane Function)** over N4.
* **PCF (Policy Control Function)**: Centralizes policy decisions (e.g., traffic prioritization).
* **UDM/UDR (Unified Data Management/Repository)**: Manages subscriber authentication, profiles, and policy data.
* **NWDAF (Network Data Analytics Function)**: Integrates with Prometheus for metrics scraping and analytics on UE mobility, network load, etc.

**Communication** among NFs uses service-based RESTful APIs over HTTP/2, ensuring a **highly scalable** 5G core.

We have made some modifications to the free5gc (e.g. the addition of NWDAF), which will be discussed in this document. As for the original structure of free5gc and how it works, you can refer to their website (<https://free5gc.org/>)

### 3.2 RAN Network (gNB)

The **RAN** is simulated with **UERANSIM**, which emulates gNBs handling radio resources for UEs. Communication with the core network occurs over the NG interface, split into:

* **NG-C (Control Plane)**: Links the gNB to the AMF for signaling (e.g., registration, handovers).
* **NG-U (User Plane)**: Connects the gNB to the UPF for data transmission.

Each gNB is configured with **unique identifiers** and **locations**, supporting **handover management** as UEs move. UERANSIM allows running multiple gNBs simultaneously, reflecting real-world deployments.

### 3.3 User Equipment (UE)

**UEs** (also simulated via UERANSIM) represent end-user devices capable of registration, session establishment, and data transmission. Each UE:

* Communicates over **RRC** (Radio Resource Control) with the gNB.
* Uses **N1** interface for signaling with the AMF.
* Establishes **PDU sessions** through the SMF and UPF once authenticated.

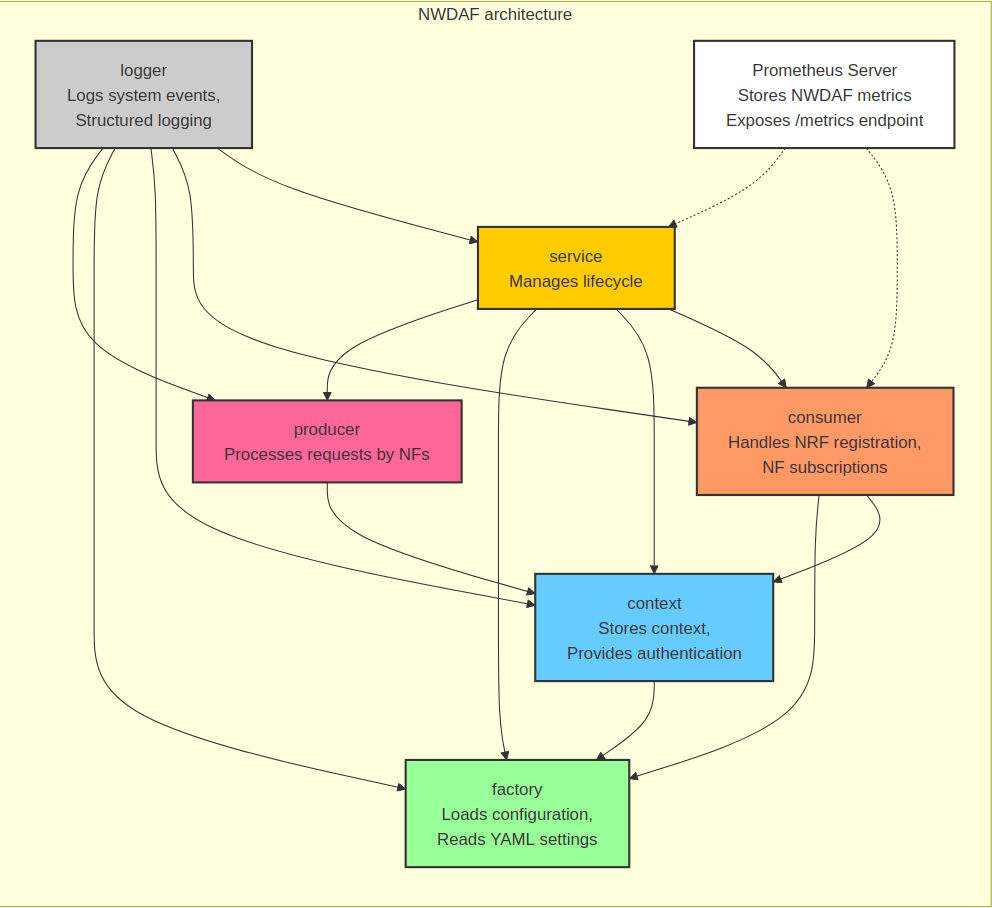
UEs have **configurable parameters** like IMSI, subscription profiles, deployment & home/work locations, enabling **realistic** testing of 5G features such as **mobility management** and **session continuity**.

## 4. Feature Deep Dives

### 4.1 NWDAF Integration

The NWDAF is a continuation of the work in (<https://github.com/net-ty/mnc_NWDAF>). Our contribution to it includes the addition of the “service” module and data collection mechanisms via subscription.

The **Network Data Analytics Function (NWDAF)** is a key analytics-driven component of the **5G Core (5GC)**, responsible for collecting, analyzing, and exposing network intelligence to optimize network performance and resource allocation. NWDAF integrates with the 5GC by first **registering itself with the Network Repository Function (NRF)** using the nnrf.go module, where it advertises its analytical services. Through NRF discovery, NWDAF identifies and subscribes to relevant **AMF and SMF events** via the amf\_subscribe.go and smf\_subscribe.go modules, enabling it to receive real-time updates on **UE registration/deregistration, mobility events, PDU session establishments/releases, and access type changes**. These event-driven notifications are processed and stored as **Prometheus metrics** within the consumer package, exposing /metrics endpoints for monitoring and visualization. The **analytics API**, implemented in api\_analytics.go, provides an external interface for NFs to request statistical insights, trend analysis, and anomaly detection via RESTful endpoints, with predictive modeling capabilities. The context.go module maintains runtime configurations, including authentication tokens and network addresses, while factory.go handles YAML-based configuration loading. The service.go module manages the entire NWDAF lifecycle, ensuring proper startup, event subscription, and controlled shutdown, including **deregistration from NRF** and **unsubscribing from NF events**. This design allows NWDAF to function as a real-time, autonomous analytics engine within the 5GC, enabling advanced network intelligence and proactive optimization of 5G network operations.

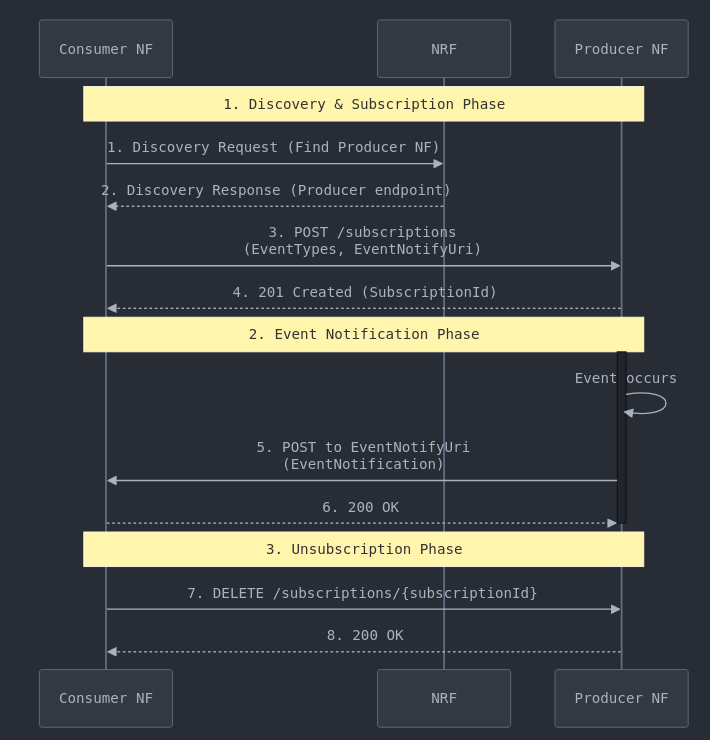


### 4.2 Event Subscription

Currently Implemented in **AMF** and **SMF**, event subscriptions let other NFs (e.g., NWDAF) receive **real-time notifications** on critical network events:

* AMF: Subscriptions for UE registration/deregistration, handovers, connectivity changes, or specific areas of interest.
* SMF: Subscriptions for session management events (establishment, release, modifications).

Events are communicated via **RESTful APIs** and reported through **HTTP callbacks**. Free5gc already had defined events for the network functions according to **3GPP** principles, and a subscription mechanism for AMF (although there wasn’t a function that sent event reports to the subscribers). We have added an event notification function for AMF and a complete subscription mechanism for SMF events.



**AMF subscription**

In the Free5GC-based testbed, Network Functions (NFs) must first query the **Network Repository Function (NRF)** to discover the **Event Exposure** service endpoint of the **Access and Mobility Management Function (AMF)** before subscribing to events, as required by the **Service-Based Architecture (SBA)**.

Originally, Free5GC supported the registration of event subscriptions in AMF but lacked a mechanism to notify subscribers when an event occurred. To address this, we implemented a **complete event notification mechanism**, allowing AMF to send event reports to subscribed NFs whenever relevant events occur.

In this testbed, **the NWDAF subscribes to AMF events** by sending a subscription request via the **Namf\_EventExposure** API. This request includes a list of event types that NWDAF wants to monitor, such as:

* **UE Registration State Changes**
* **Location Updates**
* **Presence in an Area of Interest (AOI)**
* **Time Zone Changes**
* **Access Type Reports**
* **Connectivity State Changes**
* **UE Reachability**
* **UEs in a Specific Area**

Upon receiving the subscription request, the AMF registers the subscriber in its internal **subscription store** and returns a subscription ID. The NWDAF keeps track of this ID for later **unsubscribing** when it shuts down.

To support **event notifications**, we extended Free5GC by implementing the function SendEventNotification(amfUe, eventType, status...). This function:

1. Iterates over the **AMF’s active subscriptions**.
2. Checks if the event type matches a subscriber’s requested events.
3. Constructs an **AmfEventNotification** message.
4. Sends the notification to the NF’s **EventNotifyUri**, which was provided during subscription.

Thus, when a **UE registers, moves, or changes state**, AMF automatically **sends an event notification** to NWDAF (or any other subscribed NF). This enables **real-time analytics and monitoring** of network activity.

To ensure proper lifecycle management, **NWDAF unsubscribes from AMF events when it shuts down** by invoking the **DELETE** method on its subscription ID. This prevents unnecessary active subscriptions.

By integrating this mechanism, the testbed now supports **fully functional AMF event subscriptions**, with **real-time event notifications** enhancing NWDAF’s ability to monitor network behavior dynamically.

Relevant Code Files & Functions

* **Subscription Handling (AMF)**
  + File: **api\_eventexposure.go**
    - Function: HTTPCreateSubscription(c \*gin.Context) – Handles AMF event subscription requests.
    - Function: HandleCreateAMFEventSubscription(createEventSubscription) – Stores event subscriptions in AMF.
  + File: **subscription.go**
    - Function: SendEventNotification(amfUe, eventType, status...) – Sends notifications to subscribed NFs.
    - Function: DeleteAMFEventSubscriptionProcedure(subscriptionID) – Handles unsubscriptions.
* **Subscription Handling (NWDAF)**
  + File: **amf\_subscribe.go**
    - Function: SubscribeToAMF\_UEStatus(nwdafCtx, profile) – Sends subscription requests from NWDAF.
    - Function: UnsubscribeFromAMF\_UEStatus(nwdafCtx, subscriptionId, amfProfile) – Handles NWDAF unsubscribing.

**SMF subscription**

In the Free5GC-based testbed, Network Functions (NFs) must first query the **Network Repository Function (NRF)** to obtain the **Session Management Function (SMF) Event Exposure service endpoint** before initiating a subscription request. This ensures compliance with the **Service-Based Architecture (SBA)**.

By default, Free5GC **did not support event subscriptions for SMF**. To address this limitation, we implemented a **complete event subscription mechanism**, enabling real-time monitoring of **session-related events** in the network.

In this testbed, **NWDAF subscribes to SMF events** by sending a subscription request via the **Nsmf\_EventExposure** API. This request specifies the types of events NWDAF wants to track, including:

* **PDU Session Establishment**
* **PDU Session Modification**
* **PDU Session Release**
* **UE Traffic Volume Reports**
* **UE QoS Changes**

Upon receiving a subscription request, the SMF registers the subscriber and stores the request in its **subscription list**. The function HandleCreateSMFEventSubscription() processes the request, generating a **unique subscription ID**, which is then returned to NWDAF. The NWDAF stores this **subscription ID** to later **unsubscribe when shutting down**.

To support **event notifications**, we introduced a notification mechanism in SMF:

1. **Detecting Events** – When a relevant **event** (e.g., session establishment, modification, or release) occurs, SMF checks whether any NF has subscribed to that event type.
2. **Building Event Notifications** – If a match is found, SMF constructs an NsmfEventExposureNotification message.
3. **Sending Notifications** – The event notification is then sent via **HTTP callback** to the subscriber’s EventNotifyUri (provided by NWDAF during subscription).

This mechanism enables **real-time monitoring** of session events, allowing NWDAF to analyze trends related to **session lifecycle management**, **traffic patterns**, and **network congestion**. The implementation of event notifications ensures that NWDAF can dynamically adapt to network conditions and generate analytics for decision-making.

To ensure proper **subscription lifecycle management**, NWDAF automatically **unsubscribes from SMF events** when it shuts down. It does this by invoking the **DELETE** method on its stored subscription ID, which triggers SubscriptionsSubIdDelete(). This function removes the NWDAF entry from the SMF **subscription store**, preventing orphaned subscriptions and reducing unnecessary overhead.

By implementing this functionality, the testbed now fully supports **SMF event subscriptions**, providing a **dynamic event-driven framework** that enhances **session monitoring**, **analytics**, and **network optimization** in Free5GC.

Relevant Code Files & Functions

* **Subscription Handling (SMF)**
  + File: **api\_eventexposure.go**
    - Function: SubscriptionsPost(c \*gin.Context) – Handles subscription requests for SMF events.
    - Function: HandleCreateSMFEventSubscription(c, createEventSubscription) – Registers and stores event subscriptions.
    - Function: SubscriptionsSubIdDelete(c \*gin.Context) – Handles NWDAF unsubscribing from SMF.
* **Event Notification Mechanism**
  + File: **notifier.go**
    - Function: SendEventNotification(smContext, eventType) – Sends notifications to subscribed NFs.
    - Function: SendUpPathChgEventExposureNotification(uri, notification) – Sends event exposure notifications.
* **Subscription Handling (NWDAF)**
  + File: **smf\_subscribe.go** (if applicable)
    - Function: SubscribeToSMF\_Events(nwdafCtx, profile) – Sends subscription requests from NWDAF.
    - Function: UnsubscribeFromSMF\_Events(nwdafCtx, subscriptionId, smfProfile) – Handles NWDAF unsubscribing.

### 4.3 UE Mobility

The **UE Mobility Model** implemented in udp\_task.cpp follows an **Activity-Based Mobility Model (ABMM)**, which assigns movement patterns to UEs based on time-dependent activities while ensuring realistic mobility within a defined simulation region. The primary motivation behind this model is to **simulate UE handovers** between different **gNBs (Next-Generation NodeBs)** by enabling UEs to move across multiple cells, triggering mobility-related procedures within the 5G core network. The mobility model is encapsulated in the **ActivityBasedMobilityModel** class, which defines and controls UE movement through the following key functions:

1. **initializeLocations()**
   1. Initializes a set of **predefined public locations** such as **coffee shops, restaurants, gyms, cinemas, parks, and leisure centers**.
   2. Assigns each location a **(x, y) coordinate**, a **preferred time of visit**, and a **duration** (mean and standard deviation) drawn from a Gaussian distribution.
   3. Separates **private locations**—**home and work**—which are uniquely assigned to each UE.
2. **selectNextDestination(TimeOfDay currentTime, const ActivityLocation& currentLoc)**
   1. Determines the next **destination type** based on the **current time of day** and the UE’s previous location.
   2. Uses a **probabilistic selection model** with weights assigned to each activity type.
   3. Ensures UEs do not repeatedly visit the same type of location by reducing the probability of selecting a recently visited activity.
   4. **Distinguishes between public and private locations**:
      1. **Home and work** are specific to each UE (stored in home.x, home.y, work.x, work.y).
      2. **Public locations** (restaurants, coffee shops, leisure spots) are shared across UEs.
3. **calculateRealisticSpeed(const std::string& locationType, TimeOfDay currentTime)**
   1. Generates a **realistic movement speed** based on the **UE’s current location type and time of day**.
   2. Slower speeds are assigned in areas like **work** (simulating stationary behavior), while **higher speeds** are used at night due to lower traffic.
4. **hasReachedDestination(double currentX, double currentY, double destX, double destY, double threshold = 1.0)**
   1. Checks if the UE has reached its current destination within a **specified threshold**.
   2. Triggers a state transition to **pause at the destination** before selecting the next activity.
5. **generateDuration(double mean, double stddev, std::mt19937& gen)**
   1. Uses a **Gaussian distribution** to determine how long a UE stays at its destination before selecting a new location.
6. **calculateTimeOfDay(uint64\_t currentTime)**
   1. Determines the **current time segment** (Morning, Lunch, Afternoon, Evening, Night) based on the **simulation clock**.

The **onLoop()** function orchestrates UE movement by:

1. Checking if the UE is **paused at a location**. If paused, it waits until the **pause duration** expires.
2. If the UE **needs a new destination**, it invokes **selectNextDestination()**, assigning new coordinates and adjusting movement parameters.
3. If the UE is **currently moving**, it:
   1. Calculates the **direction** and **distance** to the next destination.
   2. Moves incrementally in that direction.
   3. Calls **hasReachedDestination()** to determine if the UE has arrived.
4. When the UE **reaches a new destination**, it:
   1. Updates the **current location**.
   2. Registers a **visit event** using **Prometheus metrics** (ue\_destination\_visits\_total).
   3. Begins the **pause duration** before selecting a new destination.

Since the model operates within a **bounded simulation region** (with limits set by min\_pos, max\_pos), UEs move across **multiple gNB cells**, triggering **handover events** when:

* The UE transitions from one **cell coverage area to another**.
* A gNB observes a **signal change or loss** due to movement.
* **Inter-cell mobility** impacts connectivity, triggering **AMF and SMF event subscriptions** to track UE presence.

By dynamically changing UE locations, this model effectively facilitates testing of **handover procedures** within a 5G simulation environment, ensuring that network components (AMF, SMF, UPF) interact with mobility events in a controlled and reproducible manner.

### 4.4 Prometheus

NWDAF integrates **Prometheus** to collect, store, and expose real-time network analytics data, providing visibility into **UE registration states, mobility events, and PDU session activities**. The **consumer package** in NWDAF defines multiple Prometheus metrics, such as amf\_ue\_registration\_state, which tracks UE registration status, active\_pdu\_sessions for monitoring PDU session counts, and UE\_location\_report for recording mobility events. These metrics are updated dynamically upon receiving event notifications from AMF and SMF, where NWDAF processes **registration state changes, connectivity updates, access type changes, and PDU session establishments/releases**.

To expose these metrics, the **init.go module registers a /metrics HTTP endpoint** using promhttp.Handler(), allowing Prometheus to **scrape NWDAF metrics periodically**. Users must configure **Prometheus** by modifying prometheus.yml to include NWDAF as a target, specifying the correct IP and port, and ensuring periodic data collection. The NWDAF **analytics API** (api\_analytics.go) can further leverage collected metrics to provide **statistical insights and trend analysis**, offering external NFs a way to query analytics-based reports. Prometheus integration ensures that **NWDAF operates as a self-sustaining analytics engine**, maintaining real-time, high-fidelity data that enhances network visibility, improves decision-making, and supports **intelligent automation in 5G networks**.

Current Metrics in NWDAF

|  |  |  |
| --- | --- | --- |
| **Metric Name** | **Description** | **Labels** |
| amf\_ue\_registration\_state | Tracks UE registration state (1 = active, 0 = inactive) | supi (Subscriber ID) |
| active\_UEs | Tracks the number of currently active UEs | state |
| UE\_location\_report | Logs UE location reports | supi, tac, NrCellId, time |
| active\_pdu\_sessions | Tracks the number of active PDU sessions | State |
| total\_pdu\_session\_events | Counts total PDU session events | supi, PDU\_ID, Est\_or\_Rel |

By integrating **Prometheus**, NWDAF ensures that **network analytics are collected in real-time and made available for both operational monitoring and advanced analytics**, enabling **proactive 5G network optimization**.

To query data from prometheus in python, Users can use **PrometheusConnect**

from prometheus\_api\_client import PrometheusConnect

prom = PrometheusConnect(url=PROMETHEUS\_URL, disable\_ssl=True)

# Perform a range query using the supplied arguments

response = prom.custom\_query\_range( query=’metric name’, start\_time=start time, end\_time=end time, step='10m' # You can adjust the interval as needed

)

## 5. Installation and Configuration Guide

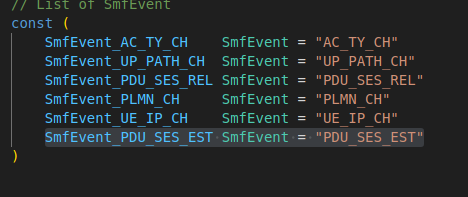
First clone this github link (<https://github.com/HenokDanielbfg/testbed>)

### 5.1 Free 5GC

* Go to this link (<https://free5gc.org/guide/3-install-free5gc/> ) and from section A (prerequisites), do all the steps.
* Once cloned, do the following

Open [/home/testbed/go/pkg/mod/github.com/free5gc/openapi@v1.0.8/models/model\_smf\_event.go](mailto:/home/testbed/go/pkg/mod/github.com/free5gc/openapi@v1.0.8/models/model_smf_event.go) and add “SmfEvent\_PDU\_SES\_EST SmfEvent = "PDU\_SES\_EST"

” to the const variables.



Then,



* Then

cd gtp5g # use version 0.86<=v<0.9.0

make

sudo make install

* Then from the link in step 1, do all of section D (Install WebConsole)

For section C, when cloning gtp5g, use version 0.86<=v<0.9.0, otherwise you might get error when you run the core  
this error to be exact, [ERRO][UPF][Main] UPF Cli Run Error: open Gtp5g: version mismatch: gtp5g version(0.9.5) should be 0.8.6 <= verion < 0.9.0 , please update it  
if you still get the error, do the following  
--> sudo rmmod gtp5g

Then reinstall gtp5g again

* Now that you have installed free5gc, run the following command on the free5gc directory ---> “./run.sh”. If it runs without any errors, then the installation is successful
* To subscribe UEs to the core, run webconsole  
  --> cd ~/free5gc/webconsole  
  --> ./bin/webconsole

This will run the webconsole on the address http://<your-local-address>:5000  
Open the link and login using “admin” for username and “free5gc” for password. Then go to subscribers and add new subscribers. Just change the supi (so its unique) and **change the ‘operator code type’ to OP**. Subscribe 5 new UEs with SUPI values 208930000000001 – 208930000000005. For more information, check out section 4 of this link (<https://free5gc.org/guide/5-install-ueransim/#4-use-webconsole-to-add-an-ue> )

After subscribing the UEs, you can terminate the webcosole using ctrl+c.

### 5.2 UERANSIM

* Run the following commands

sudo apt install make g++ libsctp-dev lksctp-tools iproute2 sudo snap install cmake --classic # Make sure version > 3.17.0

* Run this commands   
  cd UERANSIM   
  git checkout e4c492d  
  make (make sure prometheus-cpp is installed before this)
* Now UERANSIM is installed.
* In this link (<https://free5gc.org/guide/5-install-ueransim/#4-use-webconsole-to-add-an-ue>), do section 4. This is to subscribe UEs to the 5g core

### 5.3 Prometheus

* Run this commands  
  wget <https://github.com/prometheus/prometheus/releases/download/v2.41.0/prometheus-2.41.0.linux-amd64.tar.gz>tar -xvf prometheus-2.41.0.linux-amd64.tar.gz  
  cd prometheus-2.41.0.linux-amd64
* Now prometheus is installed. You can edit the configuration file to edit which endpoints to scrape. For this testbed, go to the testbed github you cloned and copy the prometheus configuration file there into your prometheus directory.
* To run prometheus, run this command (“sudo ./prometheus --config.file=prometheus.yml --storage.tsdb.retention.time=10y

”)

### 5.4 NWDAF

* Cd into /mnc\_NWDAF-main/NWDAF
* Run the nwdaf.go file to start the nwdaf.

### 5.5 Configuration files (.yaml/yml files)

**Core Network**: You can find the config files for the Network functions in the config directory of free5gc. They can be edited to fit your needs. The config file for nwdaf is found on the NWDAF directory

**gNB and UE**: both the GNBs and UEs have configuration files. The setup is loaded with a couple of GNBs and UEs. If you want to add more, you just need to create a new config file. Note that new UEs must be subscribed to the core first (using WebConsole), if they’re not already, before they can connect. When adding new config files, make sure to not duplicate the identifier fields (for e.g the ‘nci’ for GNBs and ‘supi’ for UEs). The IP addresses should also be unique.

To add a new GNB, you can duplicate an existing configuration file, give it a different name, and change the value of the following fields to make them unique.  
 \* nci, linkIp, ngapIp, gtpIp, the x,y coordinates

Then add the GNB ip address to the ‘gnbSearchList’ field of all the UEs config files.

To add a new UE, you can do the same as for GNB and change the value of the following fields to make them unique.

\* supi, imei(if supi is not used), the x,y,x1,y1 coordinates. The x,y is the home location of the UE and where it is initially deployed when joining while the x1,y1 is the work location of the UE.

## 6. Usage and Operation

### 6.1 Starting the Testbed

* **Core**: go to your free5gc directory and run the following command

--> “./run.sh”

go to your NWDAF directory and run the following command  
--> “go run nwdaf.go”

go to your promtheus directory and run the following command

--> “sudo prometheus --config.file=prometheus.yml --storage.tsdb.retention.time=10y”

The prometheus server will run on <http://localhost:9090>

* **RAN**: go to your UERANSIM directory and run the following command.

---> build/nr-gnb -c <path to your gnb config file>  
repeat this command for each GNB you want to run

* **UEs**: go to your UERANSIM directory and run the following command.

---> sudo build/nr-ue -c <path to your UE config file>

repeat this command for each UE you want to run

Note however if you want the UEs to join/leave the network from time to time, you can use the python script ‘join\_leavev2.py’ in the UERANSIM. It will automatically run 3 (number can be changed in the code) UEs.  
  
for UERANSIM usage, please refer to this link (<https://github.com/aligungr/UERANSIM/wiki/Usage>)

### 6.2 Common Workflows