I'm not a robot

Open5gs amf.yaml config example enabling suci

Note: As this space develops so quickly I’ve refreshed the original post from November 2021 in March 2021 with updated instructions. While 5G SA devices are still in their early stages, and 5G RAN hardware / gNodeBs are hard to come by, so today we’ll cover using UERANSIM to simulate UEs and 5G RAN, to put test calls through our 5GC. We’ll use Open5Gs for all the 5GC components, and install on any recent Ubuntu distribution. Installation is nice and easy; $ sudo apt update $ sudo apt install software-properties-common $ sudo add-apt-repository ppa:open5gs/latest $ sudo apt update $ sudo apt install open5gs The first point of contact we’ll need to talk about is the AMF, The AMF – the Access and Mobility Function is reached by the gNodeB over the N2 interface. The AMF handles our 5G NAS messaging, which is the messaging used by the UEs / Devices to request data services, manage handovers between gNodeBs when moving around the network, and authenticate to the network. By default the AMF binds to a loopback IP, which is fine if everything is running on the same box, but becomes an issue for real gNodeBs or if we’re running UERANSIM on a different machine. This means we’ll need to configure our AMF to bind to the IP of the machine it’s running on, by configuring the AMF in /etc/open5gs/amf.yaml, so we’ll change the ngap addr to bind the AMF to the machine’s IP, for me this is 10.0.1.207, ngap: - addr: 10.0.1.207 In the amf.conf there’s a number of things we can change and configure; such as the PLMN and network name, the NRF parameters, however for now we’ll keep it simple and leave everything else as default. To allow the changes to take effect, we’ll restart the Open5GS AMF service to make our changes take effect; $ sudo systemcl restart open5gs-amfd We’re using UERANSIM as our UE & RAN Simulator, so we’ll need to get it installed. I’m doing this on an Ubuntu system as we’re using Snaps. $ sudo apt update $ sudo apt upgrade $ sudo apt install make g++ libsctp-dev lksctp-tools $ iproute2 sudo snap install cmake --classic With all the prerequisites installed we’ll clone the Git repository and make everything from source; We’ll clone the Github repository, move into it and make from source. $ git clone $ cd UERANSIM $ make Now we wait for everything to compile, XKCD – Compiling Once we’ve got the software installed we’ll need to put together the basic settings. You should see these files in the /build/ directory and they should be executable. UERANSIM has two key parts, like any RAN, The first is the gNodeB, that connects to our AMF and handles subscriber traffic over our (simulated) radio link, The other is our subscribers themselves – the UEs. Both are defined and setup through config files in the config/ directory, While we’re not actually going to bring anything “on air” in the RF sense, we’ll still need to configure and start our gNodeB. All the parameters for our gNodeB are set in the config/open5gs-gnb.yaml file, Inside here we’ll need to set the the parameters of our simulated gNodeB, for us this means (unless you’ve changed the PLMN etc) just changing the Link IPs that the gNodeB binds to, and the IP of the AMFs (for me it’s 10.0.1.207) – you’ll need to substitute these IPs with your own of course. Now we should be able to start the gNodeB service and see the connection, let’s take a look, We’ll start the gNodeB service from the UERANSIM directory by running the nr-gnb service with the config file we just configured in config/open5gs-gnb.yaml $ build/nr-gnb -c config/open5gs-gnb.yaml All going well you’ll see something like: [2021-03-08 12:33:46.433] [ngap] [info] NG Setup procedure is successful And if you’re running Wireshark you should see the NG-AP (N2) traffic as well; If we tail the logs on the Open5GS AMF we should also see the connection too: So with our gNodeB “On the air” next up we’ll connect a simulated UE to our simulated gNodeB. We’ll leave the nr-gnb service running and open up a new terminal to start the UE with: $ build/nr-gnb -c config/open5gs-gnb.yaml But if you run it now, you’ll just see errors regarding the PLMN search failing, So why is this? We need to tell our UE the IP of the gNodeB (In reality the UE would scan the bands to find a gNB to serve it, but we’re simulating hre). So let’s correct this by updating the config file to point to the IP of our gNodeB, and trying again, So better but not working, we see the RRC was released with error “FIVEG_SERVICES_NOT_ALLOWED”, so why is this? A quick look at the logs on Open5Gs provides the answer, Of course, we haven’t configured the subscriber in Open5Gs’s UDM/UDR. So we’ll browse to the web interface for Open5GS HSS/UDR and add a subscriber, We’ll enter the IMSI, K key and OP key (make sure you’ve selected OPc and not OP), and save. You may notice the values match the defaults in the Open5GS Web UI, just without the spaces. So now we’ve got all this configured we can run the UE simulator again, this time as Sudo, and we should get a very different ouput; $ build/nr-gnb -c config/open5gs-gnb.yaml Now when we run it we should see the session come up, and a new NIC is present on the machine, uesimtun0, We can now run commands like Ping and Curl and by specifying our special uesimtun0 interface, and the traffic will be encapsulated in GTP and pop out the other end. More advanced functionality is in the works though, so keep an eye on the UERANSIM GitHub page and contribute code if you can, and consider supporting them on Patreon if you can’t, they’re doing great work. Specific configuration for **Open5gs **5G network core Configure the AMF (amf.yaml) with the PLMN required. Additionally, it is important to change the IP address indicating the location of the NF, as well as adapt the sd to the one used in the connector. ngap: - addr: 192.168.61.4 Configure the UPF (upf.yaml), indicating the IP address of the NF and the subnet of the N6. gtpu: - addr: 192.168.61.4 subnet: - addr: 10.46.0.1/16 Register the subscriber information through the WebUI. IMSI: 001010000000001 Subscriber Key (K): 465B5CE8 B199B49F AA5F0A2E E238A6BC Operator Key (OPc): E8ED289D EBA952E4 283B54E8 8E6183CA Activate the IPv4 and IPv6 forwarding and add the NAT rule. sudo sysctl -w net.ipv4.ip_forward=1 sudo sysctl -w net.ipv6.conf.all.forwarding=1 sudo iptables -t nat -A POSTROUTING -s 10.46.0.0/16 ! -o ogstun -j MASQUERADE sudo ip6tables -t nat -A POSTROUTING -s 2001:db8:cafe::/48 ! -o ogstun -j MASQUERADE Note that this commands are temporary. It might be useful to wrap them into a shell script that runs at the boot of the machine Create the TUN. sudo ip tuntap add name ogstun mode tun sudo ip addr add 10.46.0.1/16 dev ogstun sudo ip addr add 2001:db8:cafe::1/48 dev ogstun sudo ip link set ogstun up Add the IPs of the figure to the network interfaces (previously created to interconnect both machines). It can be done both with ifconfig or ip address, if one is not installed. In other example the addresses are: sudo ifconfig enp0s8 10.45.0.4 netmask 255.255.255.0 up sudo ip addr add 10.45.0.4/24 dev enp0s8 sudo ifconfig enp0s9 192.168.61.3 netmask 255.255.255.0 up sudo ip addr add 192.168.61.3/24 dev enp0s9 Modify the config.yaml file to match the network configuration (the example IP addresses are already written in the config file, so in this case, IP addresses can be matched with the ones shown in the figure). # Network Functions amf_ngap: 192.168.61.4 amf_port: 38412 upf_port: 2152 gnb_gtp: 192.168.61.3 gnbg_port: 2152 gnb_ngap: 192.168.61.3 gnbn_port: 9487 Adapt the UE and gNB configuration to match with the Open5GS ones. # UE initial_imsi: "001010000000001" mcc: "001" mnc: "01" # GNB data gnb_id: "\x00\x01\x02" gnb_bitlength: 24 gnb_name: "open5gs" # UE Authentication Data k: "465B5CE8B199B49FAA5F0A2EE238A6BC" opc: "E8ED289DEBA952E4283B54E88E6183CA" op: "E8ED289DEBA952E4283B54E88E6183CA" sst: 1 sd: "010203" Configure MAC addresses and network interfaces. src_iface is the interface of the STGUTG VM that faces the UE VM (enp0s8 in the example). dst_iface is the interface of the STGUTG VM that faces the Open5GS VM on the user plane (enp0s9 in the example). ue_number is the number of UEs to be emulated (it depends on the number of VMs deployed as UEs). # Interaces for traffic src_iface: "enp0s8" dst_iface: "enp0s9" # Number of UEs to use in traffic mode ue_number: 1 Open5Gs-UI The project utilizes Open5GS, a GNodeB, and User Equipments (UEs) to emulate a complete 5G system. GNodeB and UEs can be simulated using UERANSIM. While both can be set up locally, this project enhances simulation realism by installing Open5GS and UERANSIM on distinct virtual machines. The base OS for both machines are Ubuntu/Debian. VirtualBox is used to create the virtual machines. To enable communication between these machines, a Network Manager needs to be established. Virtual Box Network Manager configuration Upon establishing a Host-Network using the Network Manager, the network needs to be assigned as a network adapter to both machines. The Open5GS machine necessitates a secondary adapter for this purpose, whereas the UERANSIM machine only requires this single adapter as it should not access the internet through the default adapter. Open5GS network adapter settings UERANSIM network adapter settings To begin, essential developer tools like git, curl, and others need to be installed. Open5GS necessitates MongoDB, while the Open5GS-webUI demands Node.js. After configuring all dependencies, Open5GS can be installed using the apt-packet-manager. Install Open5GS and components (except webUI) sudo apt install -y gnupg curl ca-certificates curl software-properties-common git # MongoDB curl -fsSL | sudo gpg -o /usr/share/keyrings/mongodb-server-7.0.gpg --dearmor echo "deb [arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg] jammy/mongodb-org/7.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list sudo apt-get install -y mongodb-org sudo systemctl start mongod sudo systemctl enable mongod # NodeJS sudo mkdir -p /etc/apt/keyrings curl -fsSL | sudo -E bash - &&\ sudo apt-get install -y nodejs # Open5gs sudo add-apt-repository ppa:open5gs/latest -y sudo apt-get -y update && apt install -y open5gs # restart the system sudo restart Installing Open5GS might temporarily disrupt network connectivity. While restarting the system isn’t strictly mandatory, it’s the simplest way to refresh and update the operating system. After rebooting, the Open5GS-webUI can be installed and accessed. Install webUI and retrieve address # Open5gs - WebUI curl -fsSL | sudo -E bash - # Check of of WebUI journalctl -u open5gs-webui.service | grep 'Ready on' # example output $ Jan 15 07:19:04 open5g-VirtualBox node[3531]: > Ready on Upon accessing the webUI via a web browser, a new subscriber must be provisioned. For a new subscriber, only the SUPI from the GNodeB configuration file is required. Once the entire setup is complete and the system is running, the Open5GS amf.yaml and upf.yaml configuration files require editing. By default, these files are located in the /etc/open5gs/ directory. 1ngap: 2 server: 3 - address: 192.168.56.6 In the above extract from the amf.yaml the ngap-server-address needs to be replaced with the IP-Address assigned to the machine by the Host-Network. 1gtpu: 2 server: 3 - address: 192.168.56.6 In the above extract from the upf.yaml the gtpu-server-address needs to be replaced with the IP-Address assigned to the machine by the Host-Network. At this stage the Client App can be installed and run. To commence, fundamental developer tools like Git and gcc need to be installed. UERANSIM necessitates a more recent CMake version than the apt-packet-manager offers. Consequently, version 3.27.9 requires manual compilation and configuration. Install UERANSIM and components CMAKE_VERSION="3.27.9" # Setup System sudo apt update sudo apt upgrade sudo apt install make gcc g++ libsctp-dev lksctp-tools iproute2 libssl-dev git -y # Install cmake (APT version is too old!) function install_cmake () { wget -qO- CMAKE_VERSION/cmake-$CMAKE_VERSION.tar.gz | tar xvz -C ~/Downloads/ cd ~/Downloads/cmake-$CMAKE_VERSION/ || return ./configure --prefix=/opt/cmake gmake -j4 sudo make -j4 install rm -rf ~/Downloads/cmake-$CMAKE_VERSION } install_cmake # Export CMAKE-Path to environment sudo echo "$(echo "PATH=\"")"$(cat /etc/environment|grep -oP 'PATH="\K[^"]+')""$(echo ":/opt/cmake/bin")""$(echo "\"")" > /etc/environment export PATH=$PATH:/opt/cmake/bin # Install Uransim git clone ~/UERANSIM ~/UERANSIM/make -j4 # reboot system sudo reboot The UERANSIM configuration files reside in the ~/UERANSIM/config/ directory. To facilitate consistent reuse, copying and renaming the default files custom-gnb.yaml and custom-ue.yaml to gnb1.yaml and ue1.yaml is recommended. cp custom-gnb.yaml gnb1.yaml cp custom-ue.yaml ue1.yaml 1ngapIp: 192.168.56.4 # gNB's local IP address for N2 Interface (Host-adapter IP-Address) 2gtpIp: 192.168.56.4 # gNB's local IP address for N3 Interface (Host-adapter IP-Address) 3 4# List of AMF address information 5amfConfigs: 6 - address: 192.168.56.6 # Open5GS IP-Address given by Host-Adapter 7 port: 38412 The UEconfig ~/UERANSIM/config/custom-ue.yaml contains the supi, which is required to setup a subscriber with the Open5GS webUI. 1# IMSI number of the UE. IMSI = [MCC|MNC|MSISDN|] (In total 15 digits) 2supi: 'imsi-28601000000001' 1~/UERANSIM/build/nr-gnb -c ~/UERANSIM/config/gnb1.yaml 2sudo ~/UERANSIM/build/nr-ue -c ~/UERANSIM/config/ue1.yaml 3 4# UE output 5[2024-01-15 10:12:56.809] [app] [info] Connection setup for PDU session[1] is successful, TUN interface[uesimtun0, 10.45.0.3] is up. The UERANSIM nr-ue tool emits numerous messages to the console. As evident from the extract above, it records the IP address of the uesimtun0 interface, which can then be employed to establish an internet connection through Open5GS. In order to resolve any IP-Address a default gateway needs to be employed as show in the example below. sudo ip r add default dev uesimtun0 ~/UERANSIM/build/nr-binder 10.45.0.3 firefox Once everything is setup, the UERANSIM nr-binder tool can be used with many networking applications, using the tunnel interface. The client application is developed using Python 3.12, which is not available through the apt-package manager at the time of writing. Therefore, the official Python ppa:deadsnake/ppa repository is added to provide access to this version. Additionally, the application requires root privileges to capture network traffic. To achieve this, the command in line 20 launches a new shell and executes the application with root permissions. Downloading and starting the client-application PYTHON_VERSION=3.12 # clone app mkdir ~/client_app git clone ~/client_app # Installing Libpcap-dev sudo apt install -y libpcap-dev # Installing python sudo add-apt-repository ppa:deadsnakes/ppa -y sudo apt update sudo apt install -y python$PYTHON_VERSION python$PYTHON_VERSION-venv # create virtual environment python$PYTHON_VERSION -m venv ~/client_app/venv source ~/client_app/venv/bin/activate pip install --upgrade pip # Installing packages python -m pip install regex libpcap argparse pandas pip install -e ~/client_app # Network traffic can only be captured executing the application as root-user APP_DIR=/home/$(whoami)/client_app/app sudo bash -c "cd ${APP_DIR}; source ../venv/bin/activate; python main.py" Direction Source_ip Source_Host Destination_ip Dest_Host Size 0 UP 10.45.0.4 Unknown Host 142.251.37.10 muc11s23-i... 364 bytes 1 UP 10.45.0.4 Unknown Host 34.107.243.93 93.243.107... 1 KB 2 UP 10.45.0.4 Unknown Host 34.149.100.209 209.100.14... 100 bytes 3 UP 10.45.0.4 Unknown Host 8.8.8.8 dns.google 557 bytes 4 Down 142.251.37.10 muc11s23-i... 10.45.0.4 Unknown Host 1 KB 5 Down 34.107.243.93 93.243.107... 10.45.0.4 Unknown Host 4 KB 6 Down 34.149.100.209 209.100.14... 10.45.0.4 Unknown Host 1 KB 7 Down 8.8.8.8 dns.google 10.45.0.4 Unknown Host 1 KB Run with n12 In this note, I will talk about a case where we use components from multiple different product to complete a whole 5G core. In the example shown in this note, we use AUS from open5gs and all other components from Amarisoft AMF. TestSetup Configuration Log Analysis TestSetup The overall network structure of 5G core is shown as below. Among these, the components with label (A) or (O) are the components that are used in my test. I used Amari callbox as gNB and most part of 5G Core, but only AUSF of the corenetwork is using Open5GS. Test Setup for this note is shown below. I would not describe any details on setup the virtual machine and Open5GS installation. If you are not familiar with this part, refer to this note. For now, I assume that the readers are already familiar with Virtual Machine setup and Open5GS installation. Configuration Now you have to change all the necessary configuration files according to your test setup. At the initial test, I need to change only one file ausf.amf file and all other files remain as default. Following is the contents of amf.yaml and upf.yaml and the red part is what I have changed from the original configuration. Following is the configuration changes in upf ausf.yaml ausf: sbi: - addr: 192.168.100.32 # This is IP address of the PC where open5gs is installed port: 7777 # This is the port number of ausf Restart the Services Once you completed the change of configuration files, you need to restart the services which are using the changed configuration files as shown below. $ sudo systemctl restart open5gs-ausfd Log Analysis I did simple initial attach and ping from open5gs to UE. Followings are the trace log captured by Amarisoft gNB and openGS. Following is the log captured on Amarisoft Callbox WebGUI. I would not go into the details of each message since most of the message details are same as the one described in this note. The only message I will put the detailed content are N12 related messages marked as below. [3] POST From: MME Info: 192.168.100.17:9000, v2022-06-08 Time: 21:32:54.241 Index: 41 Message: 192.168.100.32:7777 POST Data: Stream id: 1 HEADERS: :method: POST :path: /nausf-auth/v1/ue-authentications :scheme: http :authority: 192.168.100.32:7777 accept: application/3gppHal+json accept: application/problem+json content-type: application/json DATA: {"supiOrSuci":"suci-0-001-01-0-0-0-0000000001","servingNetworkName":"5G:mnc001.mcc001.3gppnetwork.org"} [4] 192.168.100.32:7777 Status: 201 From: MME Info: 192.168.100.17:9000, v2022-06-08 Time: 21:32:54.264 Index: 42 Message: 192.168.100.32:7777 Status: 201 Data: Stream id: 1 HEADERS: :status: 201 server: Open5GS v2.4.7 date: Sat, 11 Jun 2022 01:32:54 GMT content-length: 318 location: content-type: application/3gppHal+json DATA: { "authType": "5G_AKA", "5gAuthData": { "rand": "be4104975fbf4db60393ce0b75ea34bf", "hxresStar": "407731e6819e9b6431c6091a006a8a56", "autn": "d69ee87c1b79900118d00077a3e5e3d0" }, "_links": { "5g-aka": { "href": " } } } [5] PUT From: MME Info: 192.168.100.17:9000, v2022-06-08 Time: 21:32:54.281 Index: 47 Message: 192.168.100.32:7777 PUT Data: Stream id: 3 HEADERS: :method: PUT :path: /nausf-auth/v1/ue-authentications/1/5g-aka-confirmation :scheme: http :authority: 192.168.100.32:7777 accept: application/json accept: application/problem+json content-type: application/json DATA: {"resStar":"1d489ca1704604dad60b958c38a296f8"} [6] 192.168.100.32:7777 Status: 200 From: MME Info: 192.168.100.17:9000, v2022-06-08 Time: 21:32:54.293 Index: 48 Message: 192.168.100.32:7777 Status: 200 Data: Stream id: 3 HEADERS: :status: 200 server: Open5GS v2.4.7 date: Sat, 11 Jun 2022 01:32:54 GMT content-length: 154 content-type: application/json DATA: { "authResult": "AUTHENTICATION_SUCCESS", "supi": "imsi-001010000000001", "kseaf": "4a221cd200c5a5e3dadb31b2a79d261896907ed0ce11f076853cde5072bcbd8c" }