OAI Reference Architecture for 5G and 6G Research with USRP

Contents

Neel Pandeya, Bharat Agarwal, Gerardo Trevino

AN-956

This Application Note presents a reference design for using the Eurecom OpenAirInterface (OAI) software stack to implement 5G NR Stand-Alone (SA) systems on the USRP N300, N310, N320, N321, X410 radio devices. The reference design covers the base station (gNB), the user equipment (UE), and the Core Network (CN) components of the network. Three types of UE are used: a UE running on a USRP radio; a wireless modem module UE;

and a commercial (COTS) handset/phone. The reference design supports operation in Frequency Range 1 (FR1), and support for operation in FR2 will be added at a future date. The various aspects of installing, configuring, and running the hardware and software components of the network are discussed in detail, along with a discussion of expected results, methods to benchmark and monitor the system, and troubleshooting steps.

The solution brochure for the OAI Reference Architecture for 5G and 6G Research with the USRP can be downloaded here.

An overview of using OAI Software for 5G and 6G research at this webpage here.

You can learn more about other solutions for 5G and 6G Wireless Research and Prototyping at the webpage here.

The USRP N300 and N310 are a networked software defined radio that provides reliability and fault-tolerance for deployment in large scale and distributed wireless systems. This device simplifies control and management of a network of radios by introducing the unique capability to remotely perform tasks such as debugging, updating software, rebooting, factory resetting, self-testing, and monitoring system health. The USRP N310 is an all-in-one device that includes two AD9371 transceivers, the Zynq-7100 SoC baseband processor, two SFP+ ports, a built-in GPSDO module, and various other peripheral and synchronization features.

The N300 features:

- Xilinx Zynq-7035 FPGA SoC
- Dual-core ARM A9 800 MHz CPU
- 2 RX, 2TX in half-wide RU form factor
- 10 MHz ? 6 GHz extended frequency range
- Up to 100 MHz of instantaneous bandwidth per channel
- RX, TX filter bank
- 16 bit ADC, 14 bit DAC
- Configurable sample rates: 122.88, 125, and 153.6 MS/s
- Two SFP+ ports (1 GbE, 10 GbE, Aurora)
- RJ45 (1 GbE)
- 10 MHz clock reference
- PPS time reference
- Built-in GPSDO
- 1 Type A USB host port
- 1 micro-USB port (serial console, JTAG)
- Watchdog timer
- OpenEmbedded Linux
- High channel density
- Reliable and fault-tolerant deployment
- Remote management capability
- Stand-alone operation
- USRP N300 **does not** contain a Trusted Platform Module

The N310 features:

- Xilinx Zynq-7100 FPGA SoC
- Dual-core ARM A9 800 MHz CPU
- 4 RX, 4TX in half-wide RU form factor
- 10 MHz ? 6 GHz extended frequency range
- Up to 100 MHz of instantaneous bandwidth per channel
- RX, TX filter bank
- 16 bit ADC, 14 bit DAC
- Configurable sample rates: 122.88, 125, and 153.6 MS/s
- Two SFP+ ports (1 GbE, 10 GbE, Aurora)
- RJ45 (1 GbE)
- 10 MHz clock reference
- PPS time reference
- External RX, TX LO input ports
- Built-in GPSDO
- 1 Type A USB host port
- 1 micro-USB port (serial console, JTAG)

- Trusted Platform Module (TPM) v1.2
- Watchdog timer
- OpenEmbedded Linux
- High channel density
- Reliable and fault-tolerant deployment
- Remote management capability
- Stand-alone operation

For more detailed technical information about the USRP N300 and N310, please reference the N300/N310 Hardware Resource Page.

For more information about getting started with the USRP N300 and N310, please reference the Getting Started Guide.

The USRP N320 and N321 are a networked software defined radio that provides reliability and fault-tolerance for deployment in large scale and distributed wireless systems. This is a high performance SDR that uses a unique RF design by Ettus Research to provide 2 RX and 2 TX channels in a half-wide RU form factor. Each channel provides up to 200 MHz of instantaneous bandwidth, and covers a frequency range from 3 MHz to 6 GHz. The baseband processor uses the Xilinx Zynq-7100 SoC to deliver a large user programmable FPGA for real-time, low latency processing and a dual-core ARM CPU for stand-alone operation.Support for 1 GbE, 10 GbE, and Aurora interfaces over two SFP+ ports and 1 QSFP+ port enables high throughput IQ streaming to a host PC or FPGA coprocessor. A flexible synchronization architecture with support for LO sharing for TX and RX, 10 MHz clock reference, PPS time reference, GPSDO, and White Rabbit enables implementation of phase coherent MIMO testbeds. The USRP N320 leverages recent software developments in UHD to simplify control and management of multiple devices over the network with the unique capability to remotely administrate tasks such as debugging, updating software, rebooting, resetting to factory state, and monitoring system health.

The USRP N320 features:

- Xilinx Zynq-7100 FPGA SoC
- Dual-core ARM A9 800 MHz CPU
- 2 RX, 2 TX in half-wide RU form factor
- 3 MHz ? 6 GHz frequency range
- Up to 200 MHz of instantaneous bandwidth per channel
- Sub-octave RX, TX filter bank
- 14 bit ADC, 16 bit DAC
- Configurable sample rates: 200, 245.76, 250 MS/s
- Two SFP+ ports (1 GbE, 10 GbE, Aurora, White Rabbit)
- One QSFP+ port (2x 10Gb / Aurora)
- RJ45 (1 GbE)
- 10 MHz clock reference
- PPS time reference
- External RX, TX LO input ports
- Built-in GPSDO
- 1 Type A USB host port
- 1 micro-USB port (serial console, JTAG)
- Trusted Platform Module (TPM) v1.2
- Watchdog timer
- OpenEmbedded Linux
- Reliable and fault-tolerant deployment
- Remote management capability
- Stand-alone operation

The USRP N321 features:

- Xilinx Zynq-7100 FPGA SoC
- Dual-core ARM A9 800 MHz CPU
- 2 RX, 2 TX in half-wide RU form factor
- 3 MHz ? 6 GHz frequency range
- Up to 200 MHz of instantaneous bandwidth per channel
- Sub-octave RX, TX filter bank
- 14 bit ADC, 16 bit DAC

- Configurable sample rates: 200, 245.76, 250 MS/s
- Two SFP+ ports (1 GbE, 10 GbE, Aurora, White Rabbit)
- One QSFP+ port (2x 10Gb / Aurora)
- RJ45 (1 GbE)
- 10 MHz clock reference
- PPS time reference
- External RX, TX LO input ports
- LO Distribution for up to 128x128 MIMO
- Built-in GPSDO
- 1 Type A USB host port
- 1 micro-USB port (serial console, JTAG)
- Trusted Platform Module (TPM) v1.2
- Watchdog timer
- OpenEmbedded Linux
- Reliable and fault-tolerant deployment
- Remote management capability
- Stand-alone operation

For more detailed technical information about the USRP N320 and N321, please reference the N320/N321 Hardware Resource Page.

For more information about getting started with the USRP N320 and N321, please reference the Getting Started Guide.

The USRP X410 is a high-performance, multi-channel software defined radio. The SDR is designed for frequencies from 1 MHz to 7.2 GHz, tunable up to 8 GHz and features a two-stage superheterodyne architecture with 4 independent TX and RX channels capable of 400 MHz of instantaneous bandwidth each. Digital interfaces for data offload and control include two QSFP28 interfaces capable of 100 GbE[1], a PCIe Gen3 x8 [3] interface, as well standard command, control, and debug interfaces: USB-C JTAG, USB-C console, Ethernet 10/100/1000. The USRP X410 is an all-in-one device built on the Xilinx Zynq Ultrascale+ ZU28DR RF System on Chip (RFSoC) with built-in digital up and down conversion and onboard Soft-Decision Forward Error Correction (SD-FEC) IP.

The USRP X410 features:

- High channel density
- Reliable and fault-tolerant deployment
- Stand-alone (embedded) or host-based (network streaming) operation
- Fully integrated and assembled (the USRP X410 does not support swappable daughtercards)
- 1 MHz to 7.2 GHz frequency range (tunable up to 8GHz)
- Up to 400 MHz of instantaneous bandwidth per channel
- 4 RX, 4 TX in half-wide RU form factor
- Xilinx Zynq-Ultrascale+ ZU28DR RFSoC
- 12 bit ADC, 14 bit DAC
- IQ Sample Clock rates up to 500 MS/s
- Onboard SD-FEC, DDC, DUC
- Quad-core ARM Cortex-A53 up to 1.2 GHz CPU
- Dual-core ARM Cortex-A5 MPCore up to 500 MHz
- Two QSFP28 ports (10 Gigabit Ethernet, 100 Gigabit Ethernet, Aurora)
- Two iPass+? zHD® Interfaces (PCIe Gen3 x 8)
- RJ45 (1 GbE) [1]
- 10 MHz Clock reference
- PPS time reference
- Trig In/Out Interface
- Built-in GPSDO
- Two FPGA Programmable GPIO Interfaces (HDMI)
- 1 Type C USB host port

- 1 Type C USB port (serial console, JTAG)
- Watchdog timer
- OpenEmbedded Linux
- USRP Hardware Driver? (UHD) open-source software API version 4.1.0 or later
- RF Network on Chip (RFNoC?) FPGA development framework
- Xilinx Vivado® 2019.1 Design Suite (license not included)
- GNU Radio support maintained by Ettus Research? through GR-UHD, an interface to UHD distributed by GNU Radio
- [1] The RJ45 port is used for remote management of the device and does not support IQ streaming.

For more detailed technical information about the USRP X410, please reference the X410 Hardware Resource Page.

For more information about getting started with the USRP X410, please reference the X410 Getting Started Guide.

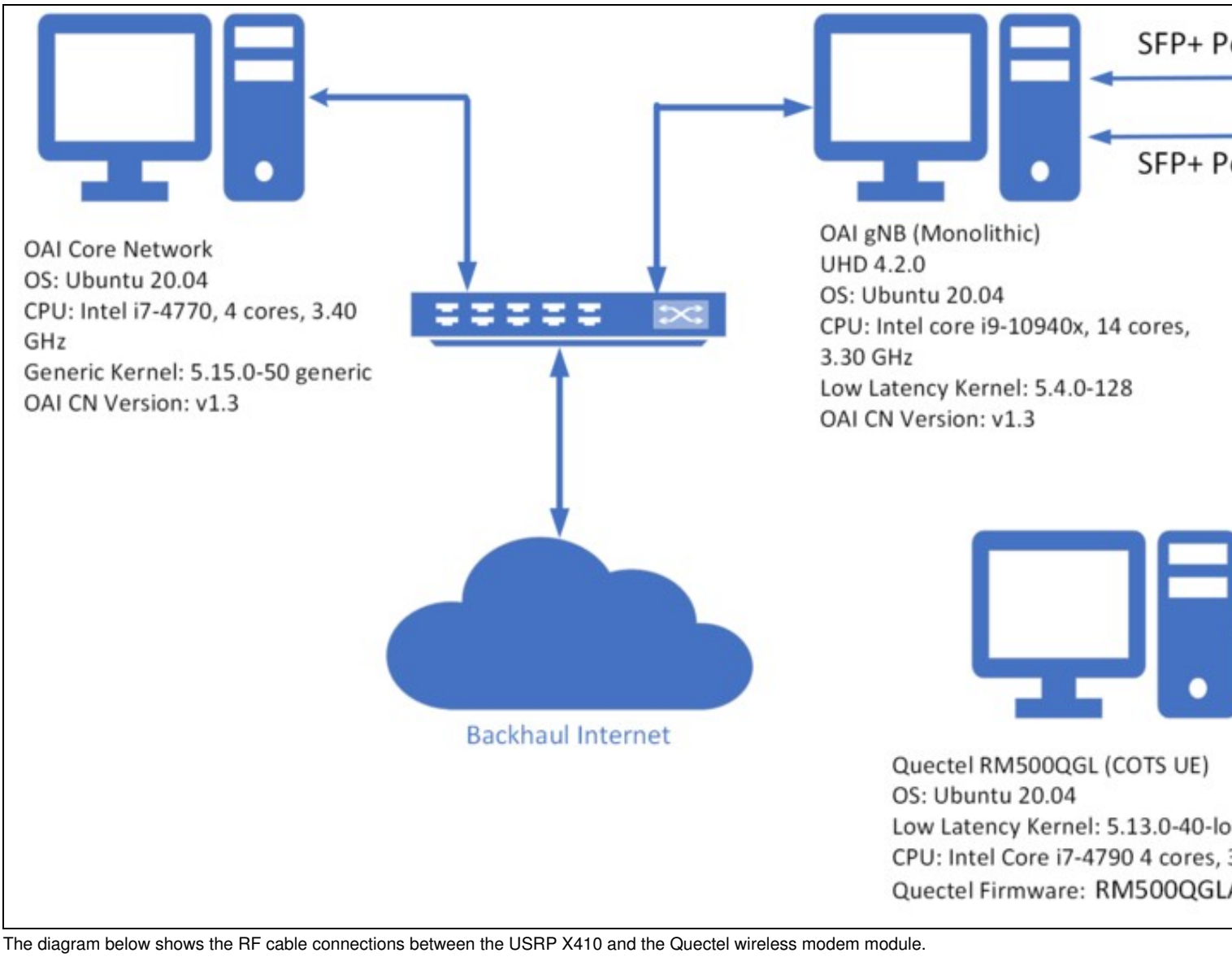
The OpenAirInterface (OAI) software provides an open-source, standards-compliant implementation of a 3GPP 5G NR stack that runs on a commodity x86 CPU and a USRP radio device. OAI was initially developed by Eurecom, which is a university in France. It is now managed and developed by the OpenAirInterface Software Alliance (OSA), which is a French non-profit organization that provides open-source software and tools for 4G and 5G wireless research. Further information can be found on the Eurecom website.

The OAI software provides a 5G NR implementation that runs in real-time and is capable of operating with commercial 5G NR handsets (UEs). The OAI software includes implementations of the gNB, the UE, and the Core Network (CN). Further information can be found on the OAI website and their GitLab repository, listed below.

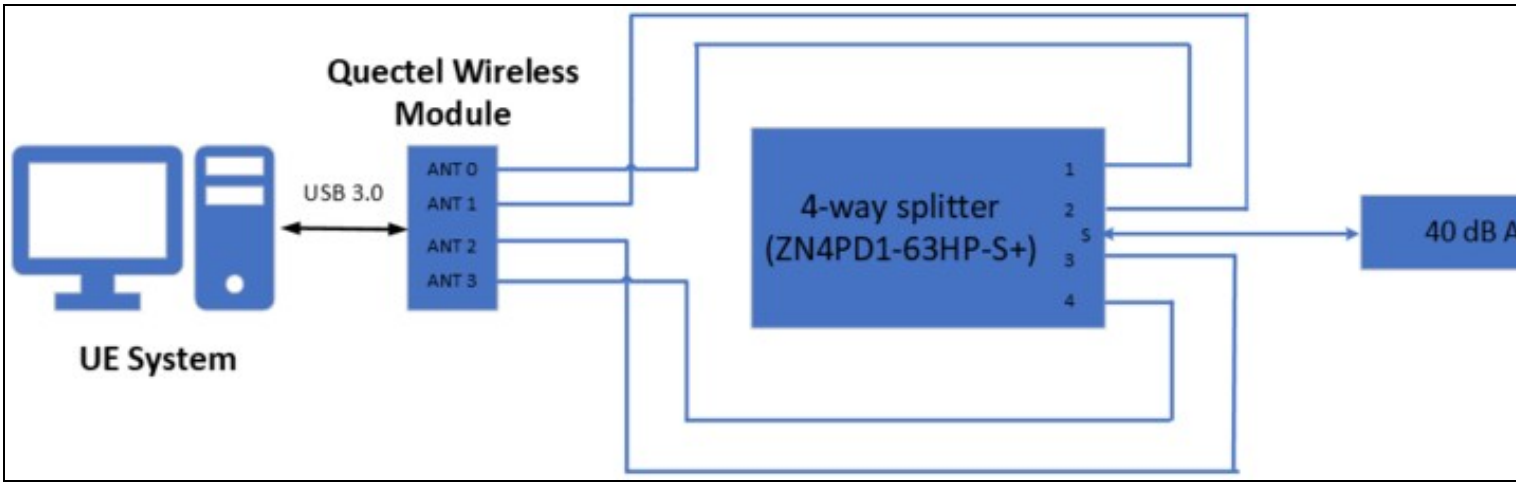
- OpenAirInterface 5G Radio Access Network Project
- OAI GitLab Repository
- OAI 5G Core Network
- OAI-CN GitLab repository

The availability of OAI source code is free for non-commercial and academic research purposes. More information about licensing can be found on the OAI website at here and here.

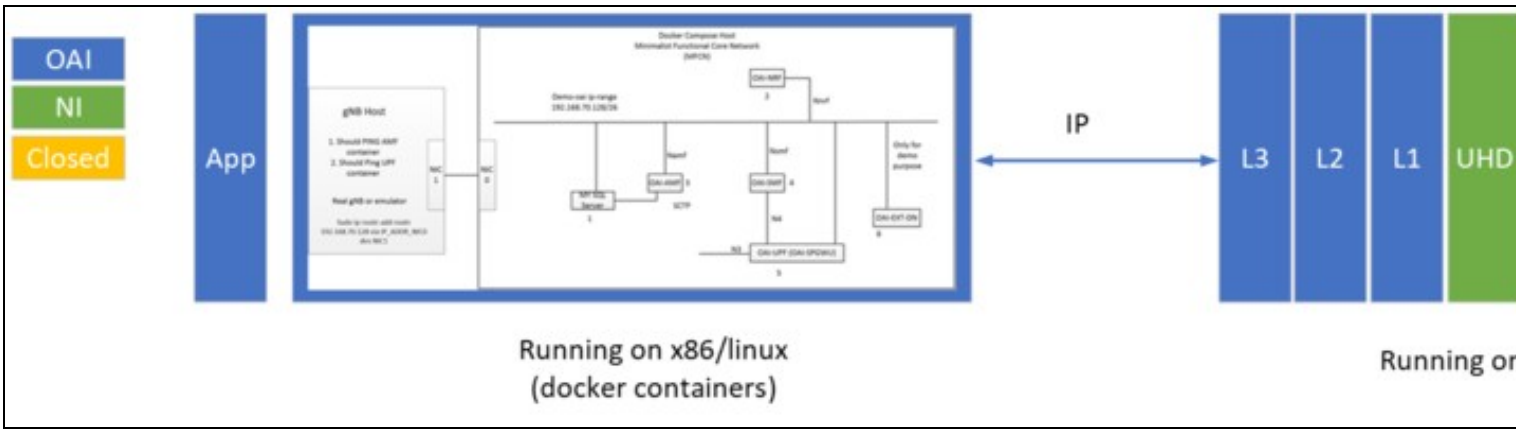
The diagram below shows the layout and architecture of this 5G OAI USRP reference architecture.



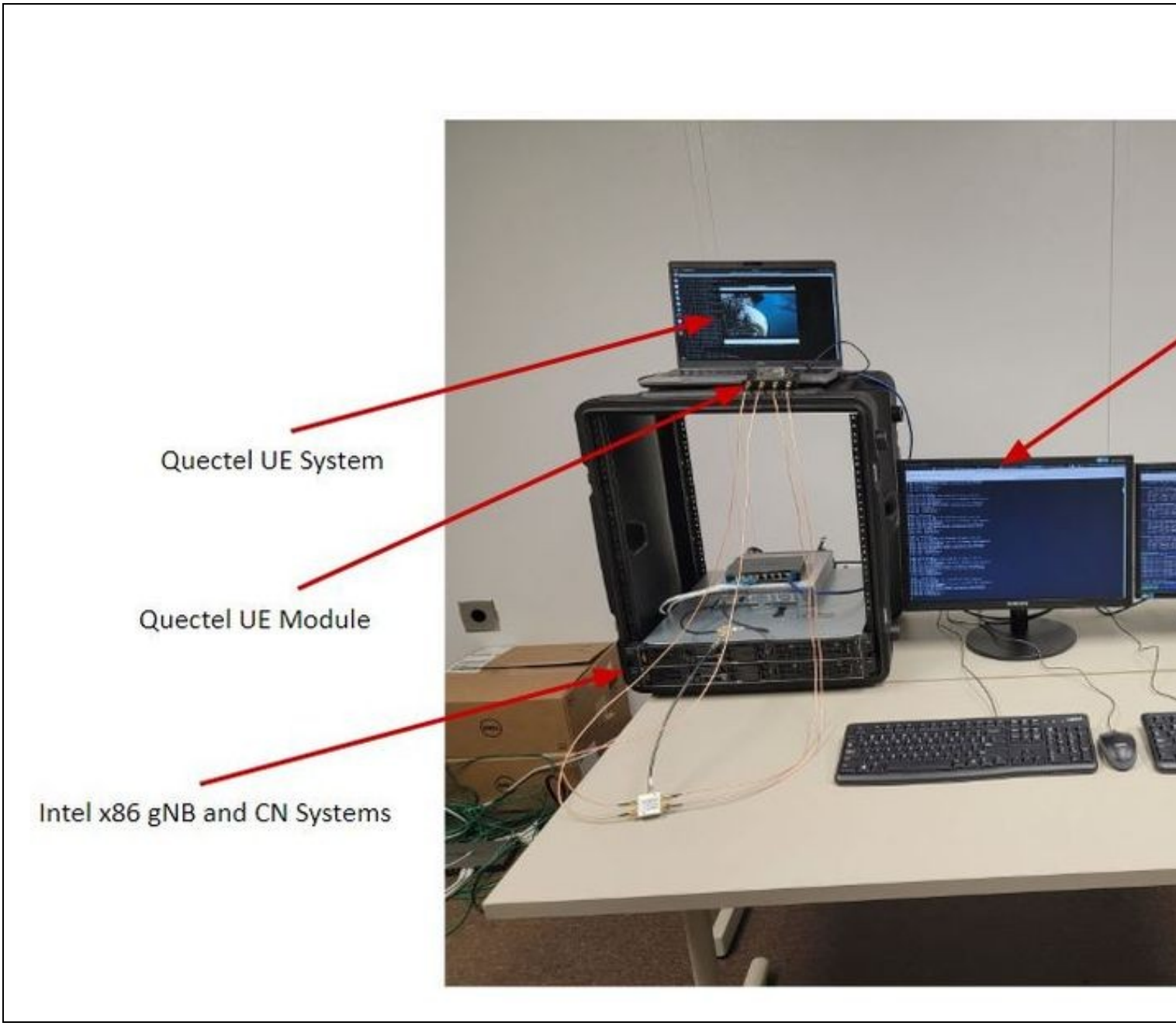
The diagram below shows the RF cable connections between the USRP X410 and the Quectel wireless modem module.

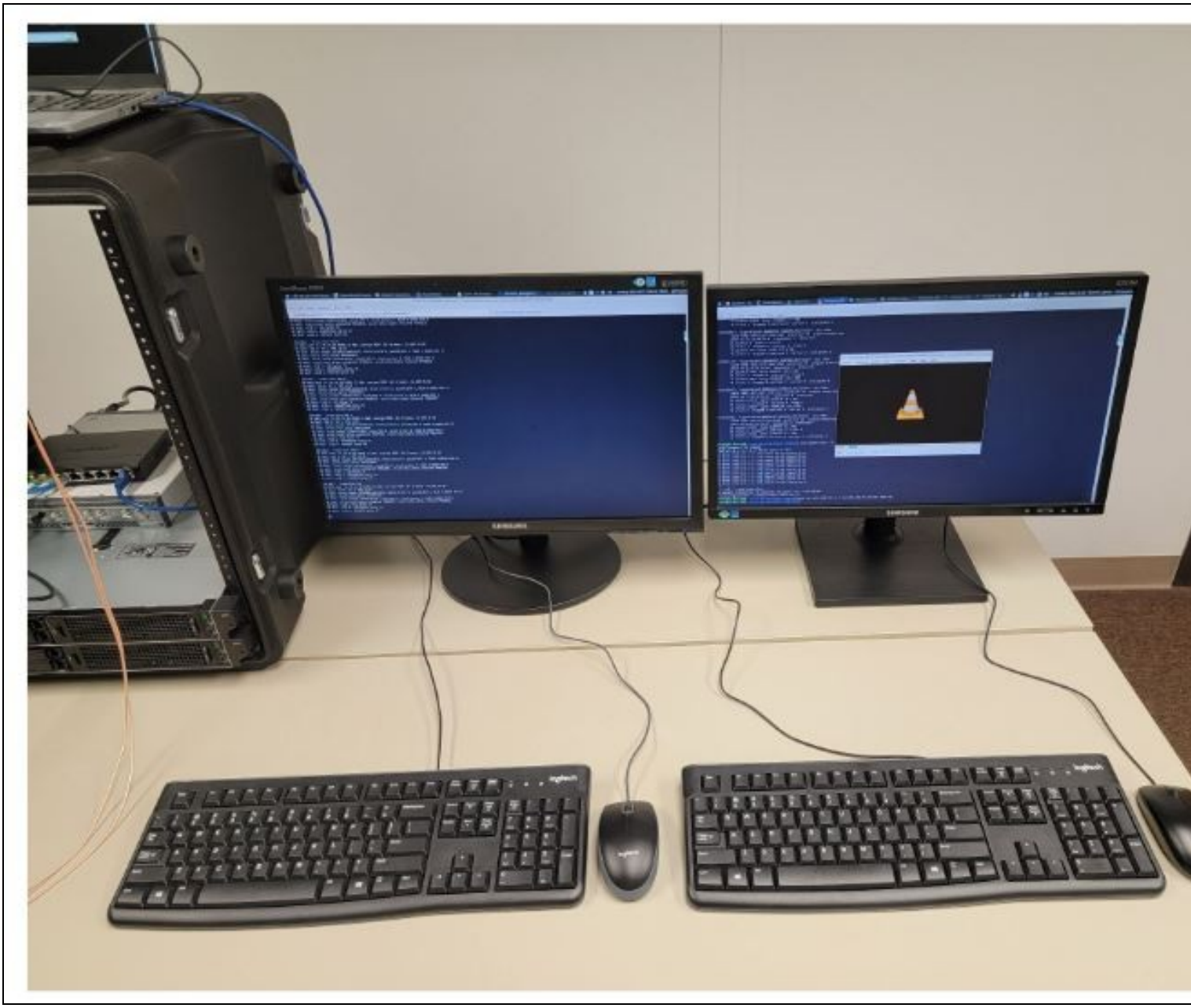


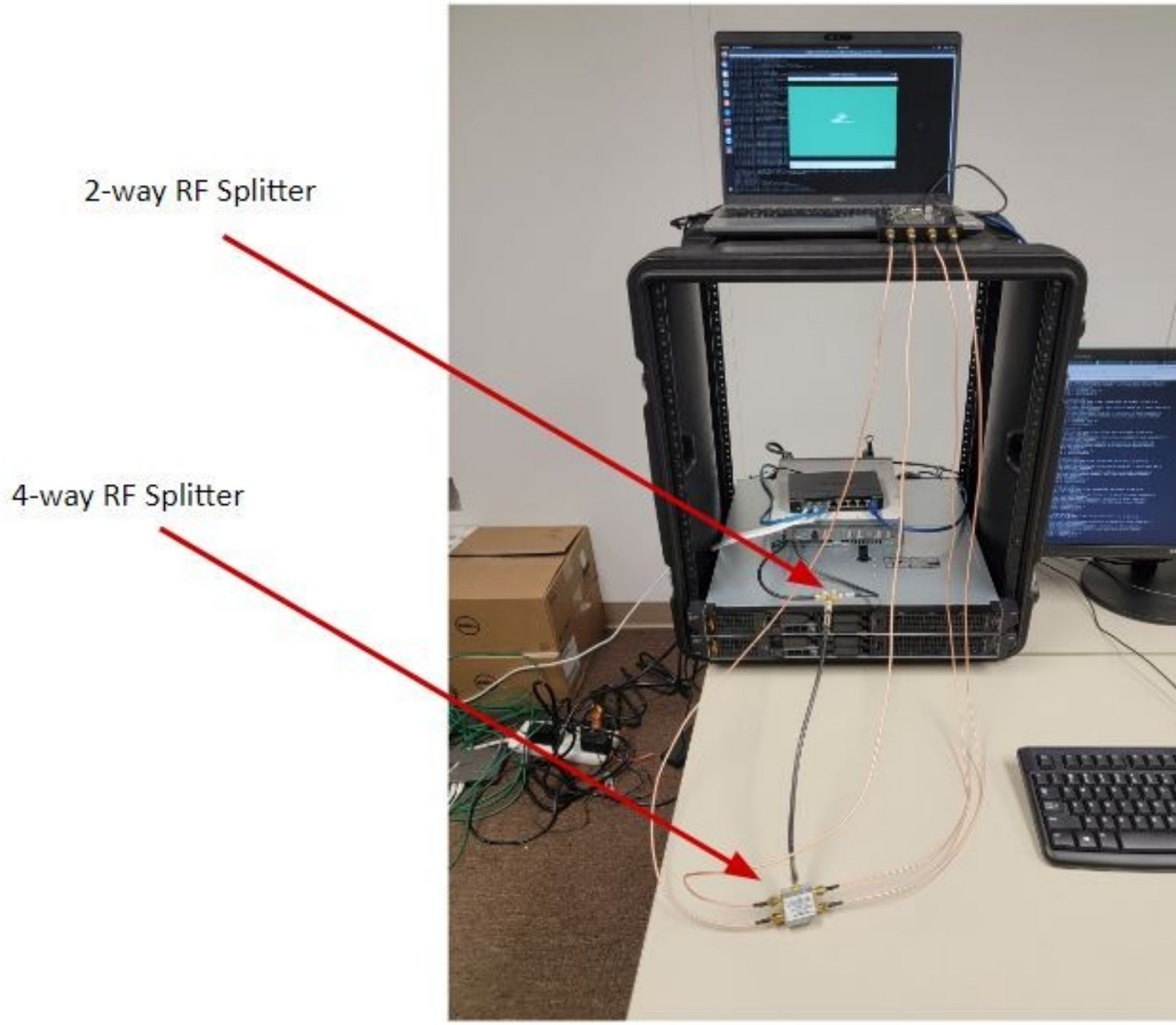
The diagram below shows the software layout and architecture across the Core Network, gNodeB, and UE systems.

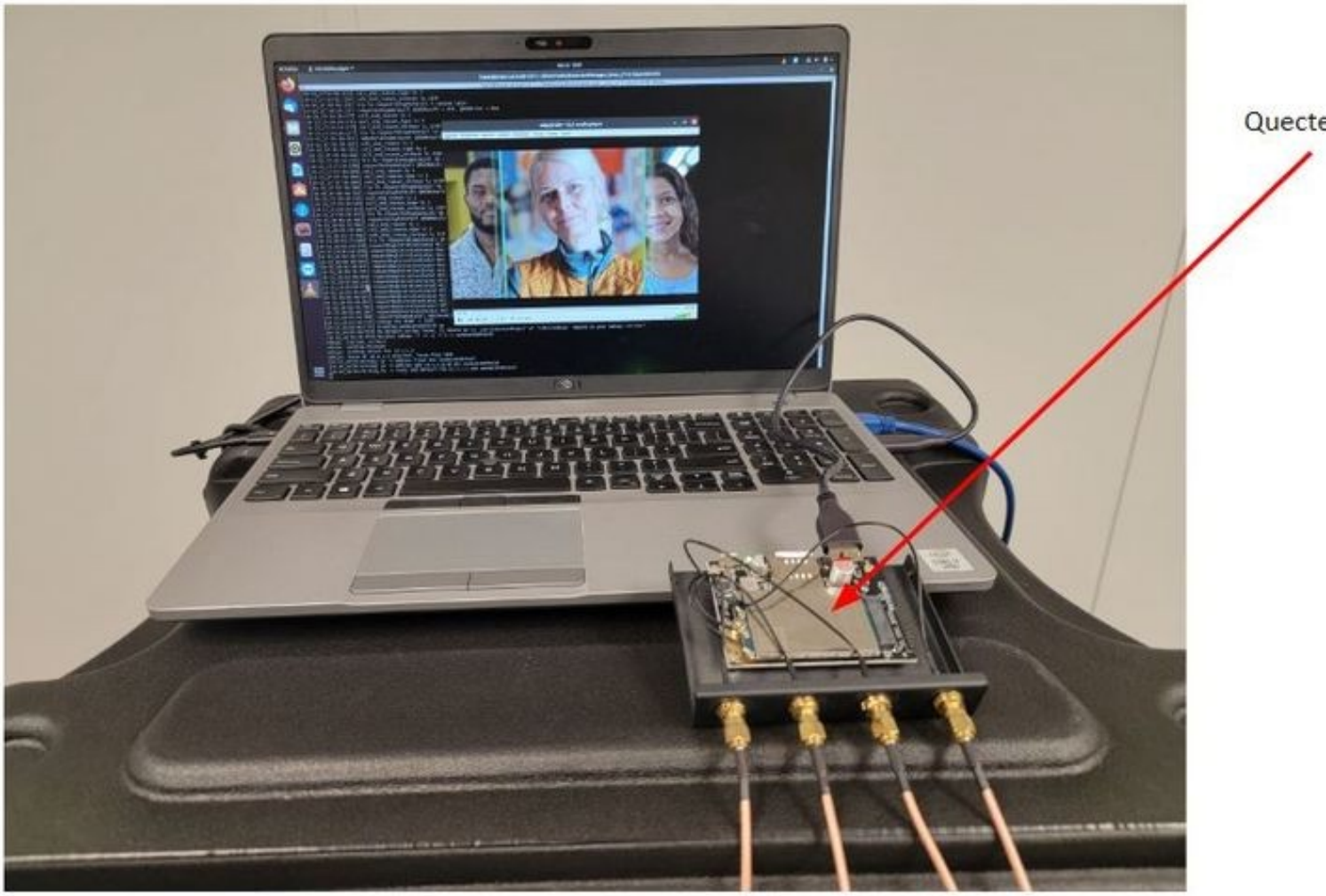


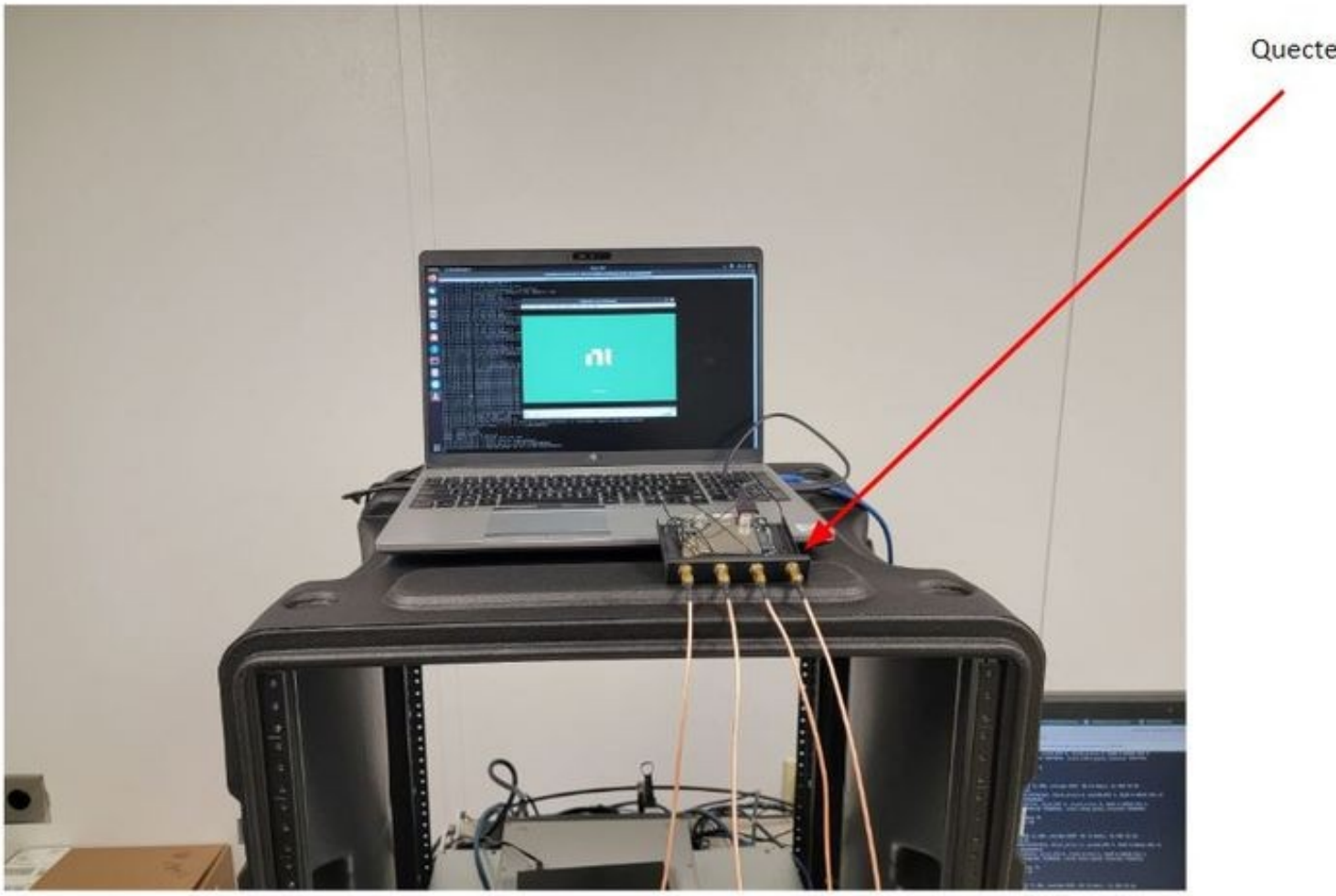
The photos below show one physical implementation of the reference architecture in the office of the authors. Note that this is not the sole valid system implementation, but it is one of many possible valid system implementations.



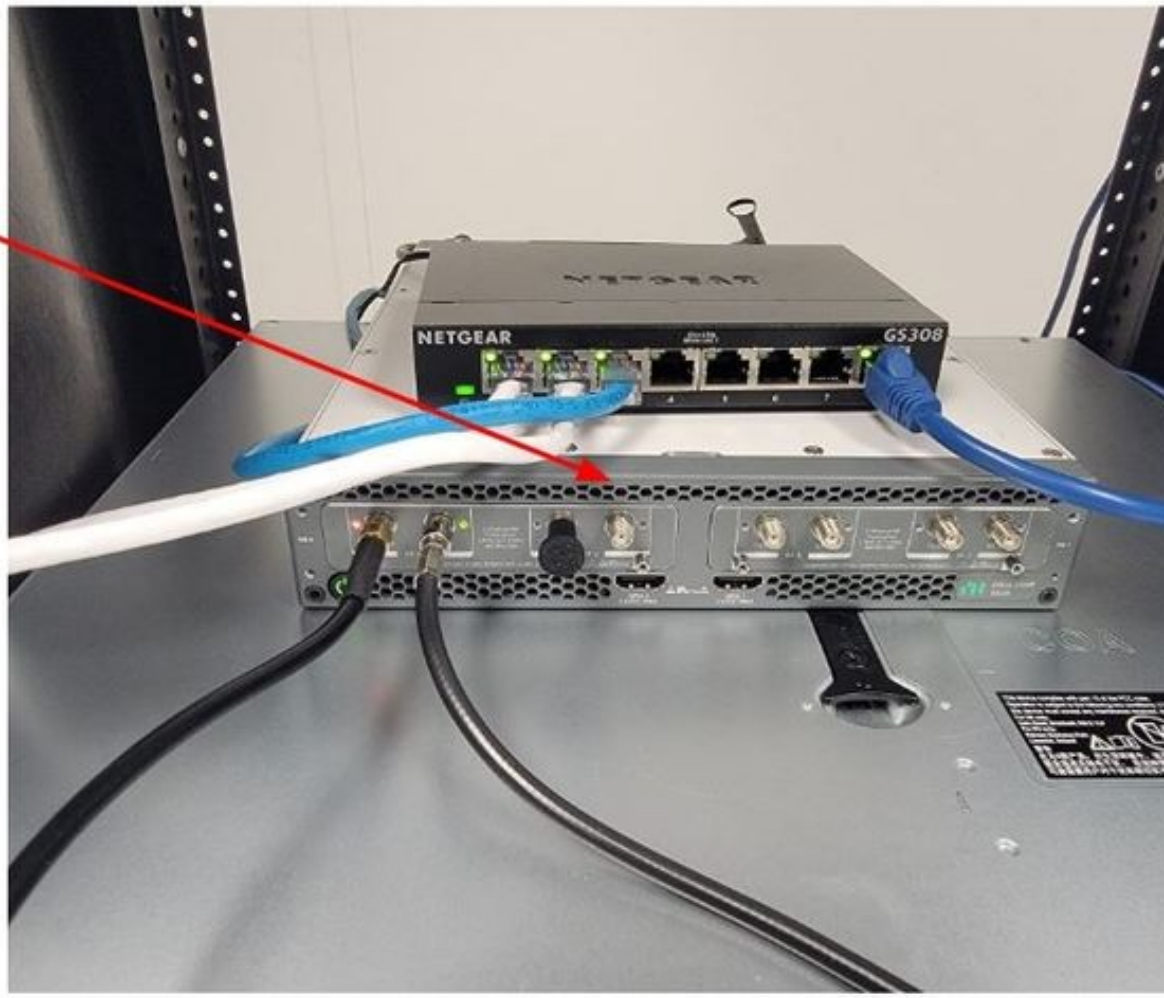



2-way RF Splitter
4-way RF Splitter


Quecte


Quecte

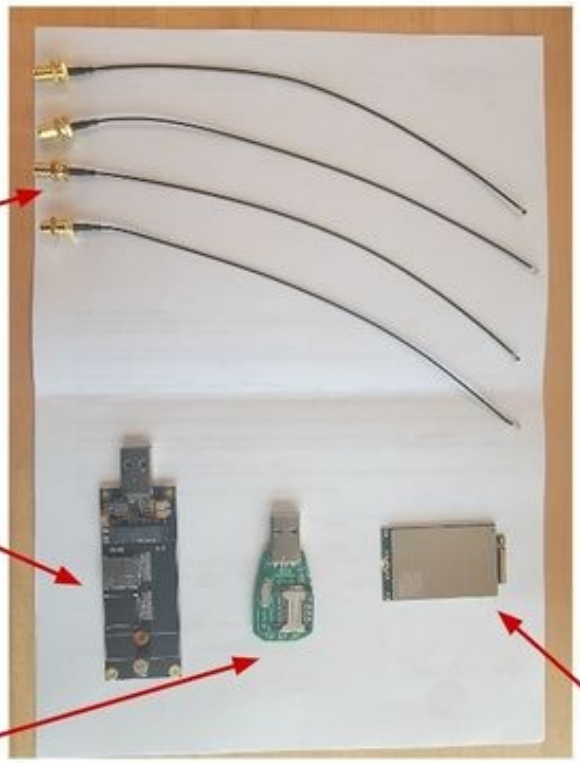
USRP X410



Pigtail MHF4-to-SMA RF Cables

USB 3.0 / M.2 Carrier Board

SIM Card
Reader/Writer



Quectel Module

Quectel Module

USB 3.0 / M.2 Carrier Board

The full Bill of Materials (BoM) for the reference design is listed below. The BoM includes all the hardware components for the multiple system configuration scenarios, where the USRP N300, N310, N320, N321, X410 can be used for the gNB and the UE system, and the UE itself can be implemented with a USRP device, or with a wireless modem module, or with a COTS handset.

- Three desktop computers, with Intel Core i9 CPU, of 10th, 11th, or 12th Generation, with clock speed of minimum 4.0 GHz, with minimum 10 physical cores, and also with only NVMe disk drives. See further details about this item in the Hardware Requirements section.
- Two 10 Gbps Ethernet networks cards. We recommend the Intel X710-DA2 and the Nvidia/Mellanox MCX4121A-ACAT ConnectX-4 network cards. See further details about this item in the Hardware Requirements section.
- Two USRP devices. The USRP may be any of USRP N300, N310, N320, N321, X410. There will be one USRP for the gNB, and one USRP for the UE. The USRP devices can be mixed (i.e., the gNB could run with a USRP X410, while the UE runs with a USRP N310).
  - https://kb.ettus.com/N300/N310
  - https://kb.ettus.com/N320/N321
  - https://kb.ettus.com/X410
  - https://www.ettus.com/all-products/usrp-n300/
  - https://www.ettus.com/all-products/usrp-n310/
  - https://www.ettus.com/all-products/usrp-n320/
  - https://www.ettus.com/all-products/usrp-n321/
  - https://www.ettus.com/all-products/usrp-x410/
- One QSFP28-to-SFP28 breakout cable (NVIDIA MCP7F00-A003R26N DAC Splitter Cable Ethernet 100GbE to 4x25GbE 3m). This is only needed when using the USRP X410.
  - https://store.mellanox.com/products/nvidia-mcp7f00-a003r26n-passive-copper-splitter-cable-ethernet-100gbe-to-4x25gbe-qsfp28-to-4xsfp28
  - https://store.mellanox.com/products/nvidia-mcp7f00-a003r30l-passive-copper-splitter-cable-ethernet-100gbe-to-4x25gbe-qsfp28-to-4xsfp28-
- One OctoClock-G. This is needed to synchronize the gNB USRP and the UE USRP. Ensure that device used is the "-G" model, which contains an internal GPSDO module. This is only needed when the UE is implemented on a USRP device.
  - https://kb.ettus.com/OctoClock_CDA-2990
  - https://www.ettus.com/all-products/octoclock-g/
- Four 10 Gbps Ethernet cables with SFP+ terminations. These cables are available in 0.5, 1.0, and 3.0 meter lengths. These cables are needed when using the USRP N300, N310, N320, N321, but not when using the USRP X410.
  - https://www.ettus.com/all-products/10gige-dc/
  - https://www.ettus.com/all-products/10gige-1m/
  - https://www.ettus.com/all-products/10gige-3m/
- Four VERT900 antennas and/or four VERT2450 antennas. You can use either of these antennas based on the bands in use. There are also many third-party vendors selling a wide variety of antennas, such as omnidirectional antennas. As long as the antenna has an 50-ohm impedance and has SMA connectors, then it can be used with the USRP radio.
  - https://www.ettus.com/all-products/vert900/
  - https://www.ettus.com/all-products/vert2450/
- One Quectel RM500Q-GL 5G wireless modem module. See further details about this item in the Hardware Requirements section.
  - https://www.quectel.com/product/5g-rm500q-gl
  - https://www.quectel.com/product/5g-rm50xq-series
- One Google Pixel 5A 5G handset (phone). Be sure that the handset is unlocked.
  - https://www.gsmarena.com/google_pixel_5-10386.php
  - https://www.gsmarena.com/google_pixel_5a_5g-11059.php
  - https://www.amazon.com/Google-Pixel-5G-Factory-Unlocked/dp/B09DV93S9K/

- Two 5G SIM cards and one USB UICC/SIM card reader/writer.
  - https://open-cells.com/index.php/sim-cards/
- One Mini-Circuits 4-way DC-Pass, SMA, Power Splitter, 250 to 6000 MHz, 50? (ZN4PD1-63HP-S+).
  - https://www.minicircuits.com/WebStore/Splitters.html
  - https://www.minicircuits.com/WebStore/dashboard.html?model=ZN4PD1-63HP-S%2B
- Two Mini-Circuits, 2-way, DC-Pass SMA, Power Splitter, 500 to 5000 MHz, 50? (ZN2PD2-50-S+).
  - https://www.minicircuits.com/WebStore/Splitters.html
  - https://www.minicircuits.com/WebStore/dashboard.html?model=ZN2PD2-50-S%2B
- Four Mini-Circuits, VAT-10+, 10 dB Fixed Attenuator, DC to 6000 MHz, 50?, SMA.
  - https://www.minicircuits.com/WebStore/dashboard.html?model=VAT-10%2B
- Four Mini-Circuits, VAT-20+, 20 dB Fixed Attenuator, DC to 6000 MHz, 50?, SMA.
  - https://www.minicircuits.com/WebStore/dashboard.html?model=VAT-20%2B
- Four Mini-Circuits, VAT-30+, 30 dB Fixed Attenuator, DC to 6000 MHz, 50?, SMA.
  - https://www.minicircuits.com/WebStore/dashboard.html?model=VAT-30%2B
- Fourteen Coax SMA Cables, Mini-Circuits, SMA, Hand-Flex Interconnect, 18.0 GHz, 36 inches length (086-36SM+).
  - https://www.minicircuits.com/WebStore/dashboard.html?model=086-36SM%2B
  - https://www.minicircuits.com/pdfs/086-36SM+.pdf
- One NETGEAR GS108 8-Port Gigabit Ethernet Unmanaged Switch. See further details about this item in the Hardware Requirements section.
  - https://www.netgear.com/business/wired/switches/unmanaged/gs108/
  - https://www.amazon.com/NETGEAR-Ethernet-Unmanaged-Lifetime-Protection/dp/B00MPVR50A/
- Three adapters from USB 3.0 to 1 Gbps Ethernet. Any major reputable brand will work. Use either USB-A or USB-C connector as per the ports available on your host computer.
  - https://www.amazon.com/Network-Adapter-CableCreation-Ethernet-Supporting/dp/B013G4C8RE/
  - https://www.amazon.com/Ethernet-Thunderbolt-Gigabit-Network-Compatible/dp/B07XTGKP5M/

Three host computers are needed, one for the gNB, one for the UE, and one for the CN, with the specifications discussed in this section. The requirements for the host computer running the CN are not as high as for the gNB and UE, but we recommend that all three host computers meet the requirements described here. We also strongly recommend that each of the gNB, UE, and CN be implemented on their own dedicated system. A single host computer should only run the gNB, or the UE, or the CN.

We recommend using an Intel Core i9 CPU, or an Intel Xeon CPU, 10th, 11th, or 12th Generation, with minimum clock speed of 4.0 GHz, and with minimum 10 physical cores. Examples of such a CPU would be the Intel i9-10940X CPU, which has 14 physical cores, 4.60 GHz clock speed, and is 10th generation, and the Intel i9-12900K CPU, which has 16 physical cores, 5.20 GHz clock speed, and is 12th generation, as well as the Intel Xeon Platinum 8351N. Be sure that the CPU has at least 40 PCIe lanes, or at least enough lanes to support any GPU and 10 Gbps Ethernet card that are being used. The system should ideally support PCIe Gen 4.

- https://www.intel.com/content/www/us/en/products/sku/198014/intel-core-i910940x-xseries-processor-19-25m-cache-3-30-ghz/specifications.html
- https://ark.intel.com/content/www/us/en/ark/products/134599/intel-core-i912900k-processor-30m-cache-up-to-5-20-ghz.html

We strongly recommend that only NVMe SSD disks are used. Many systems now use PCIe Gen-4, and we recommend using a PCIe Gen-4 NVMe SSD disk. We recommend the Samsung 980 PRO SSD drive. Do not SATA disks at all. Using a RAID configuration with multiple drives should not be necessary, although this can potentially be an option for further increasing performance and throughput.

- https://www.samsung.com/us/computing/memory-storage/solid-state-drives/980-pro-pcie-4-0-nvme-ssd-1tb-mz-v8p1t0b-am/
- https://www.amazon.com/SAMSUNG-PCIe-Internal-Gaming-MZ-V8P1T0B/dp/B08GLX7TNT/
- https://www.tomshardware.com/reviews/samsung-980-pro-m-2-nvme-ssd-review

The system should have either dual-channel or quad-channel DDR4 or DDR5 (preferred) memory, with the highest clock speed available. A minimum of 16 GB or 32 GB of memory should be sufficient. Larger amounts of memory should not be necessary, as no virtualization, RAM disk, or other large in-memory buffering is being used.

The GPU does not matter for the purposes of running UHD and OAI. If you might be doing some AI/ML processing in then GPU, then you may want to use a particular GPU. The OAI 5G stack does not currently leverage the GPU.

The gNB and UE system will need a two-port 10 Gbps Ethernet network card for connecting to the USRP radio. The CN system does not connect to any USRP, so it does not need a 10 Gbps Ethernet network card.

There are several cards that are relevant and that we recommend, depending on the specific use-case.

The Intel X710-DA2 is a solid network card, and works out-of-the-box with Ubuntu 20.04.4. However, it has some issues with DPDK. This reference design is not initially using DPDK, so these issues are not yet relevant. The card is widely availabe and relatively inexpensive. The X710-DA4 is a four-port version of the card.

Be sure that the computer's chassis can physically accommodate the network card.

- https://ark.intel.com/content/www/us/en/ark/products/83964/intel-ethernet-converged-network-adapter-x710da2.html
- https://www.amazon.com/Intel-Ethernet-Converged-X710-DA2-X710DA2/dp/B00NJ3ZC26/
- https://www.cdw.com/product/intel-ethernet-converged-network-adapter-x710-da2-network-adapter-pcie/3473844
- https://www.newegg.com/intel-x710da2/p/N82E16833106253

The Mellanox MCX4121A-ACAT ConnectX-4 is also a solid network card, and also works out-of-the-box with Ubuntu 20.04.4. Furthermore, it works well with DPDK.

- https://store.mellanox.com/products/nvidia-mcx4121a-acat-connectx-4-lx-en-adapter-card-25gbe-dual-port-sfp28-pcie3-0-x8-rohs-r6.html
- https://www.mellanox.com/files/doc-2020/pb-connectx-4-lx-en-card.pdf
- https://store.nvidia.com/en-us/networking/store/product/MCX4121A-ACAT/nvidiamcx4121a-acatconnectx-4lxenadaptercard25gbe/
- https://www.amazon.com/Mellanox-ConnectX-4-MCX4121A-ACAT-25Gigabit-Ethernet/dp/B011HVAZ78/
- https://www.fs.com/products/119650.html

The USRP X410 only has a QSFP28 port, which is for 100 Gbps Ethernet. In order to connect the USRP X410 to the host computer via 10 Gbps Ethernet, a QSFP28-to-SFP28 breakout cable is needed. This cable will be required when using the USRP X410, and will connect directly to the 10 Gbps Ethernet cards. This cable is not needed for the USRP N300, N310, N320, N321.

It is certainly possible to directly connect the host computer to the 100 Gbps QSFP28 port of the USRP X410. In order to do this, a QSFP28 100 Gbps Ethernet card would be needed. We recommend the Mellanox/Nvidia MCX516A-CCAT (PCIe Gen3), and the Mellanox/Nvidia MCX516A-CDAT (PCIe Gen4). You will also need a QSFP28 cable. We recommend the Mellanox/Nvidia MCP1600-C003E26N, and the Mellanox/Nvidia MCP1600-C003E30L and MCP7F00-A001R30N. We also recommend the Intel E810 series network cards. There are both 100 Gbps Ethernet QSFP28 cards and 10 Gbps Ethernet SFP28/SFP+ cards. All these cards work well with Ubuntu 20.04 and DPDK.

However, in this release of the reference design, it is not necessary to use 100 Gbps Ethernet, which is only needed for supporting high aggregate data rates, such as for the larger 200 MHz and 400 MHz FR2 channel bandwidths, and/or for 2x2 MIMO configuration, both of which are not yet supported. For this release of the reference design, we recommend using the Intel X710-DA2 and the Mellanox MCX4121A-ACAT ConnectX-4 cards, and using dual 10 Gbps Ethernet connections.

- https://store.nvidia.com/en-us/networking/store/product/MCP7F00-A003R26N/nvidiamcp7f00-a003r26ndacsplittercableethernet100gbeto4x25gbe3m/
- https://store.nvidia.com/en-us/networking/store/product/MCP7F00-A003R30L/nvidiamcp7f00-a003r30ldacsplittercableethernet100gbeto4x25gbe3m/
- https://store.mellanox.com/products/nvidia-mcp7f00-a001r30n-passive-copper-splitter-cable-ethernet-100gbe-to-4x25gbe-qsfp28-to-4xsfp28-1m-colore
- https://store.mellanox.com/products/nvidia-mcx516a-ccat-connectx-5-en-adapter-card-100gbe-dual-port-qsfp28-pcie3-0-x16-tall-bracket-rohs-r6.html
- https://store.mellanox.com/products/nvidia-mcx516a-cdat-connectx-5-ex-en-adapter-card-100gbe-dual-port-qsfp28-pcie4-0-x16-tall-bracket-rohs-r6.htm
- https://store.mellanox.com/products/nvidia-mcp1600-c003e26n-passive-copper-cable-ethernet-100gbe-qsfp28-3m-black-26awg-ca-n.html
- https://store.mellanox.com/products/nvidia-mcp1600-c003e30l-passive-copper-cable-ethernet-100gbe-qsfp28-3m-black-30awg-ca-l.html
- https://ark.intel.com/content/www/us/en/ark/products/series/184846/100gbe-intel-ethernet-network-adapter-e810.html
- https://www.intel.com/content/www/us/en/products/details/ethernet/800-network-adapters/e810-network-adapters/products.html
- https://www.intel.com/content/www/us/en/products/details/ethernet/800-network-adapters/e810-25gbe-network-adapters/products.html

There are many vendors who sell host computers that meet these requirements. This reference design was implemented using System 76 Thelio Mira systems and Dell Precision 5820 systems. Both systems have flexible configuration options, and can be configured as described in this section.

- https://system76.com/desktops/thelio-mira-b2/configure
- https://www.dell.com/en-us/work/shop/desktops-all-in-one-pcs/precision-5820-tower-workstation/spd/precision-5820-workstation

Two USRP devices are needed, one for the gNB, and one for the UE. The USRP may be any of USRP N300, N310, N320, N321, X410. The USRP devices can be mixed (i.e., the gNB could be implemented with a USRP X410, while the UE could be implemented with a USRP N310). All these USRP devices can support all the channel bandwidths in FR1, up to and including 100 MHz. For FR2, the USRP N320 can support the 50, 100, 200 MHz channel bandwidths, and the USRP X410 can support all the 50, 100, 200, 400 MHz channel bandwidths.

- https://kb.ettus.com/N300/N310
- https://kb.ettus.com/N320/N321
- https://kb.ettus.com/X410
- https://www.ettus.com/all-products/usrp-n300/
- https://www.ettus.com/all-products/usrp-n310/
- https://www.ettus.com/all-products/usrp-n320/
- https://www.ettus.com/all-products/usrp-n321/
- https://www.ettus.com/all-products/usrp-x410/

One OctoClock-G device is needed to synchronize the gNB USRP and the UE USRP. Ensure that device used is the "-G" version of the OctoClock, which contains an internal GPSDO module. This is only needed when the UE is implemented on a USRP device.

- https://www.ettus.com/all-products/octoclock-g/
- https://kb.ettus.com/OctoClock_CDA-2990

One implementation of the UE is a 5G wireless modem module, such as the Quectel RM500Q-GL. The modem module requires a device driver and a connection to a host computer. It is possible to use one of the gNB, UE, CN host computers to also drive the modem module, but it is recommended to have a separate dedicated fourth host computer for this. This host computer need not be powerful or high-performance, and may run either Ubuntu 20.04 or Windows 10. It may be preferred to use Windows 10, as the Qualcomm drivers may work better on Windows than on Linux.

- https://www.quectel.com/product/5g-rm500q-gl
- https://www.quectel.com/product/5g-rm50xq-series

The required operating system for the gNB, UE, CN systems is Ubuntu 20.04.4. Be sure to use the Desktop image, not the Server image. It is also necessary to use the low-latency kernel on the gNB and UE systems, but not on the CN system. The kernel version should be 5.15 for Ubuntu 20.04.4. The "Installing and Configuring the UHD Software" section contains detailed information about what specific package dependencies need to be installed, and how to install the low-latency kernel. Either Kubuntu or Xubuntu may also be used, instead of Ubuntu. Do not run Ubuntu in a Virtual Machine (VM). Do not use any virtualization. Be sure to install Ubuntu on-the-metal.

The reference architecture will add support for Ubuntu 22.04.1 in the near future.

- https://releases.ubuntu.com/20.04/
- https://xubuntu.org/download/
- https://kubuntu.org/getkubuntu/
- https://en.wikipedia.org/wiki/Ubuntu
- https://en.wikipedia.org/wiki/Xubuntu
- https://en.wikipedia.org/wiki/Kubuntu

UHD is the open-source device driver for all USRP radios. The required version of UHD for this reference design is 4.2.0.0. The "Installing and Configuring the UHD Software" section contains detailed information about the installation and configuration procedure. The "Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux" Application Note also contains details and thorough information about how to install and configure UHD. We recommend that you build UHD from source code, and do not install it from a binary package. We also recommend that you build UHD first, before building and installing OAI, and that you do not build UHD using the OAI `build_oai` script. UHD is required on both the gNB system and the UE system, but it is not needed on the CN system.

- https://github.com/EttusResearch/uhd

The required version of OAI for this reference design is either 2022.w33 or the `devel` branch. We recommend that you build OAI from source code, using the `build_oai` script. The OAI software is required on the gNB, UE, and CN systems. The "Installing and Configuring the CN System", "Installing and Configuring the gNB System", and "Installing and Configuring the UE System" sections contains detailed information about the installation and configuration procedure for the OAI software. Be sure to build and install OAI only after building and installing UHD.

- https://gitlab.eurecom.fr/oai/openairinterface5g

The Data Plane Development Kit (DPDK) is an open-source software project that provides a set of data plane libraries and network interface controller polling-mode drivers for offloading TCP packet processing from the operating system kernel to processes running in user-space. This offloading achieves higher computing efficiency and higher packet throughput than is possible using the interrupt-driven processing provided in the kernel. By putting the network interface driver in user space, avoiding context switches, and pinning I/O threads to cores, UHD and DPDK combine to largely prevent the latency spikes induced by the kernel scheduler, and the overall overhead for packet processing is reduced.

The current version of the reference design does not use DPDK, but it is expected that DPDK will be required in future versions of the reference architecture.

The "Getting Started with DPDK and UHD" Application Note contains detailed information about DPDK.

- https://www.dpdk.org/
- https://github.com/DPDK/dpdk
- https://en.wikipedia.org/wiki/Data_Plane_Development_Kit

This section explains how to build and install UHD from source code. At the time of this writing, we recommend using UHD version 4.3.0.0.

Be sure to first install all the required dependencies, which can be done using the command listed below. You can run this command even if the dependencies are already installed and you just want to verify that.

```
sudo apt-get install autoconf automake build-essential ccache cmake cpufrequtils doxygen ethtool g++ git inetutils-tools libboost-all-dev
```

First start by creating a working folder to store Git repositories.

```
cd $HOME/git
mkdir $HOME/git
```

Next, clone the UHD repository on GitHub in the working folder.

```
cd $HOME/git
git clone http://github.com/EttusResearch/uhd.git
```

Then, create a build folder, and select UHD version 4.3.0.0.

```
cd uhd/host
mkdir build
cd build
git checkout v4.3.0.0
```

Then, build UHD, using the default settings, and install it to the default location.

```
cmake ../
make -j4
sudo make install
sudo ldconfig
```

Finally, add the following lines the end of your $HOME/.bashrc file.

```
export PYTHONPATH=/usr/local/lib/python3/dist-packages:/usr/local/lib/python3.7/site-packages:/usr/local/lib/python3/dist-packages:$PYTHON
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

Verify that UHD was correctly installed. Check the version number, and be sure that the uhd_find_devices utility program run.

```
uhd_config_info --print-all
uhd_find_devices
uhd_usrp_probe
```

The USRP N300, N310, N320, N321, X410 can all be used as the gNB and the UE.

See the article USRP N300/N310/N320/N321 Getting Started Guide for information about how to set up and configure your USRP N300 and N310 for use with UHD and OAI.

The USRP N300 and N310 support all the channel bandwidths in FR1.

See the article USRP N300/N310/N320/N321 Getting Started Guide for information about how to set up and configure your USRP N320 and N321 for use with UHD and OAI.

The USRP N320 and N321 support all the channel bandwidths in FR1, and all but the 400 MHz channel bandwidth in FR2.

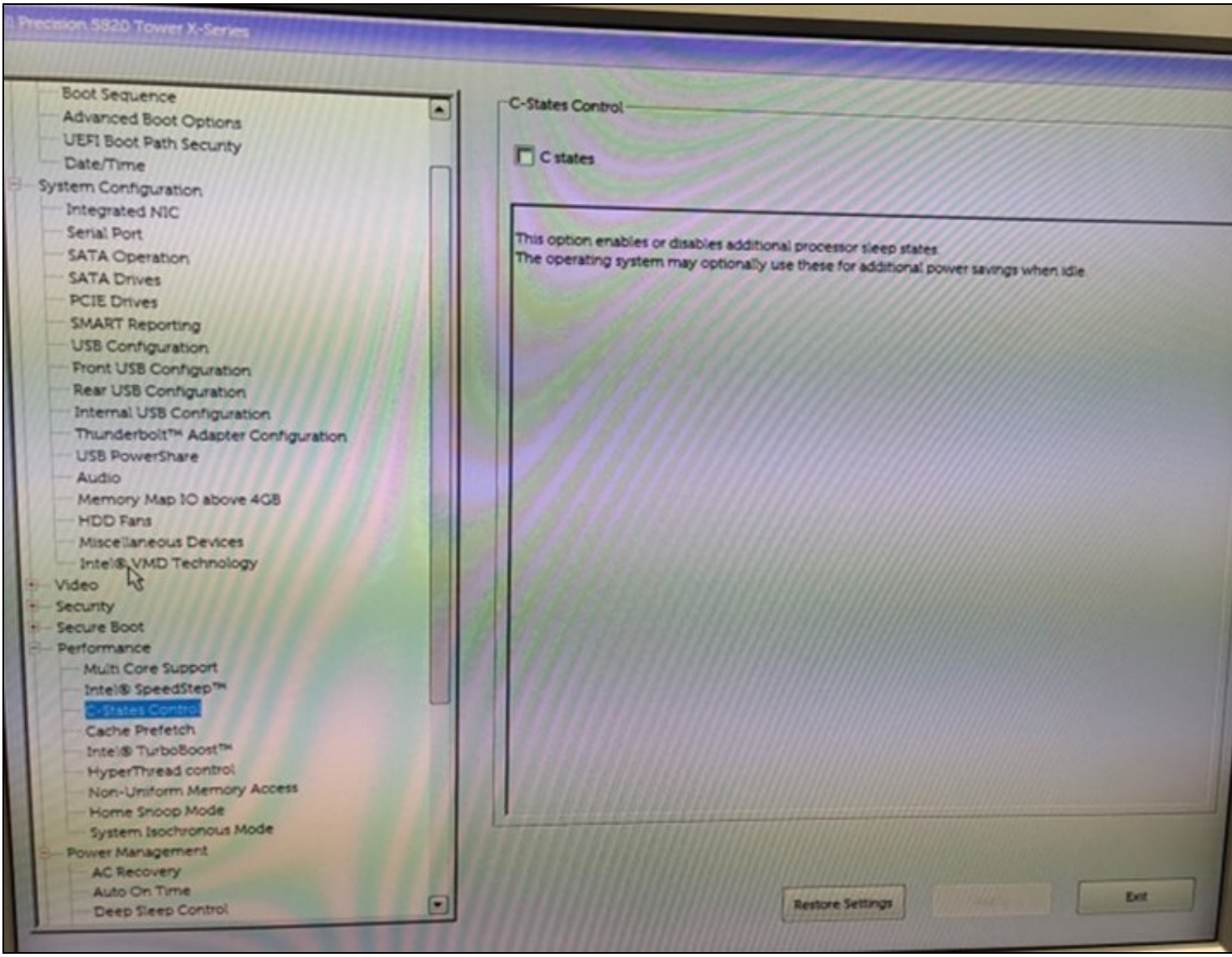
See the article USRP X410 Getting Started Guide for information about how to set up and configure your USRP X410 for use with UHD and OAI.

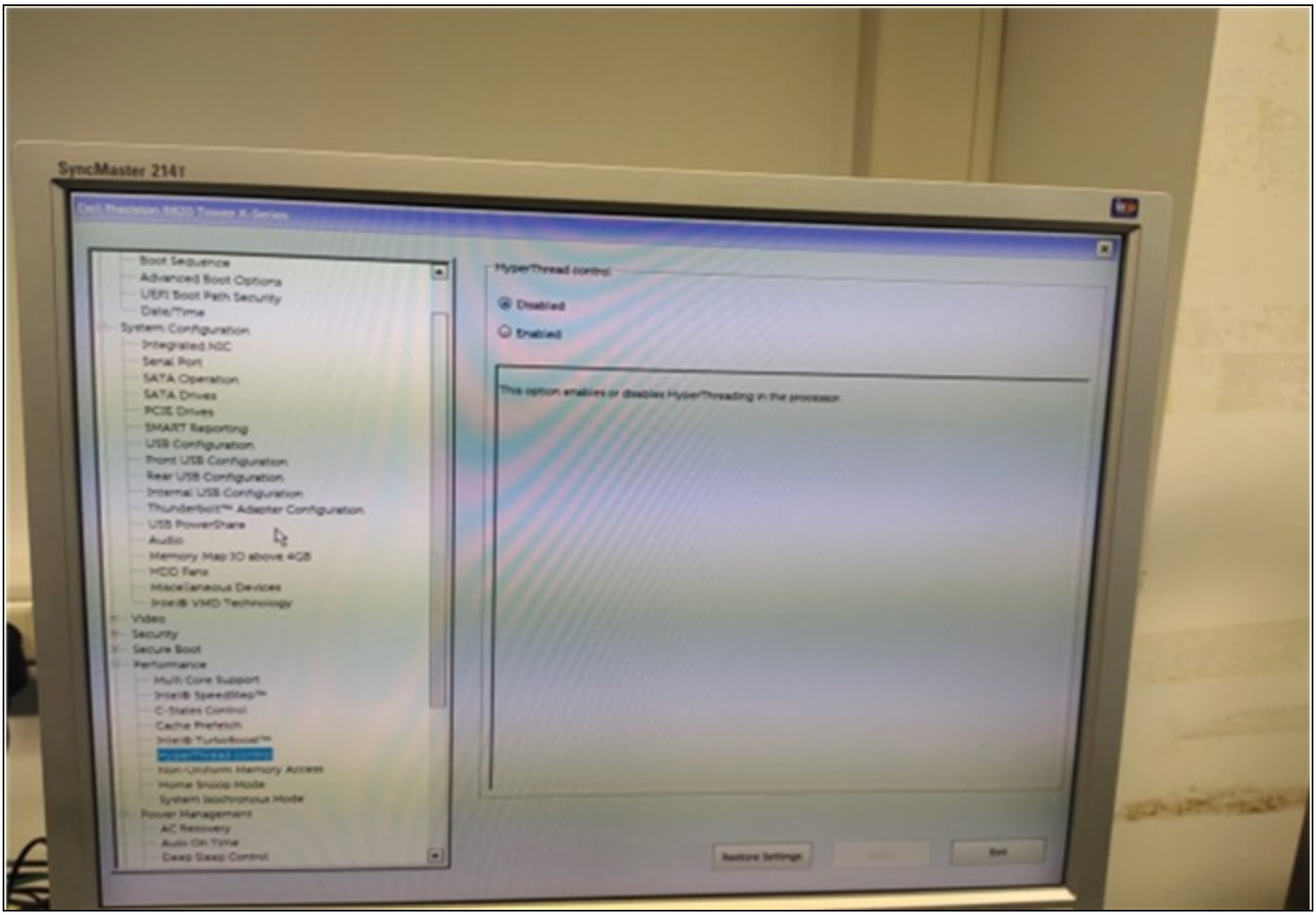
The USRP X410 supports all the channel bandwidths both in FR1 and FR2.

See the article USRP Host Performance Tuning Tips and Tricks for information about specific settings and configuration procedures needed to enable optimal system performance. Specifically, it is necessary to set the CPU governors, enable the thread priority scheduling, set the socket buffer sizes, set the Ethernet MTU values, and set the network card ring buffer sizes.

The use of the Data Plane Development Kit (DPDK) should not be necessary for running any of the FR1 channel bandwidths. At the time of this writing, DPDK is not used in this reference architecture.

On the gNB system, and on the UE system, if the UE being used is a USRP radio, there are several specific settings that need to be made. The hyperthreading, CPU frequency control, C-states, P-states, and any other power management should all be disabled. Each BIOS will have a different way to do this. The two screenshots below show where in the BIOS menu to disable C-states and to disable Hyperthreading. Once these options are set in the BIOS, reboot the system.


Precision 5820 Tower X-Series
Boot Sequence
Advanced Boot Options
UEFI Boot Path Security
Date/Time
System Configuration
Integrated NIC
Serial Port
SATA Operation
SATA Drives
PCIE Drives
SMART Reporting
USB Configuration
Front USB Configuration
Rear USB Configuration
Internal USB Configuration
Thunderbolt™ Adapter Configuration
USB PowerShare
Audio
Memory Map IO above 4GB
HDD Fans
Miscellaneous Devices
Intel® VMD Technology
Video
Security
Secure Boot
Performance
Multi Core Support
Intel® SpeedStep™
C-States Control
Cache Prefetch
Intel® TurboBoost™
HyperThread control
Non-Uniform Memory Access
Home Snoop Mode
System Isochronous Mode
Power Management
AC Recovery
Auto On Time
Deep Sleep Control
C-States Control
C states
This option enables or disables additional processor sleep states
The operating system may optionally use these for additional power savings when idle
Restore Settings
Exit



This section explains the deployment of the OAI 5G CN system on Ubuntu 20.04 with a generic kernel. The deployment is possible either using docker-compose or Helm Chart. We recommend using docker-compose.

The 5G CN deployment is possible in various forms using docker-compose or Helm Chart.

- Using docker-compose, perform a minimalist deployment
- Using docker-compose, perform a basic deployment
- Using docker-compose, perform a basic-vpp deployment with VPP implementation of UPF
- Using docker-compose, perform a basic deployment with static UE IP address allocation
- Using Helm Chart, perform a basic deployment
- Using docker-compose, doing network slicing

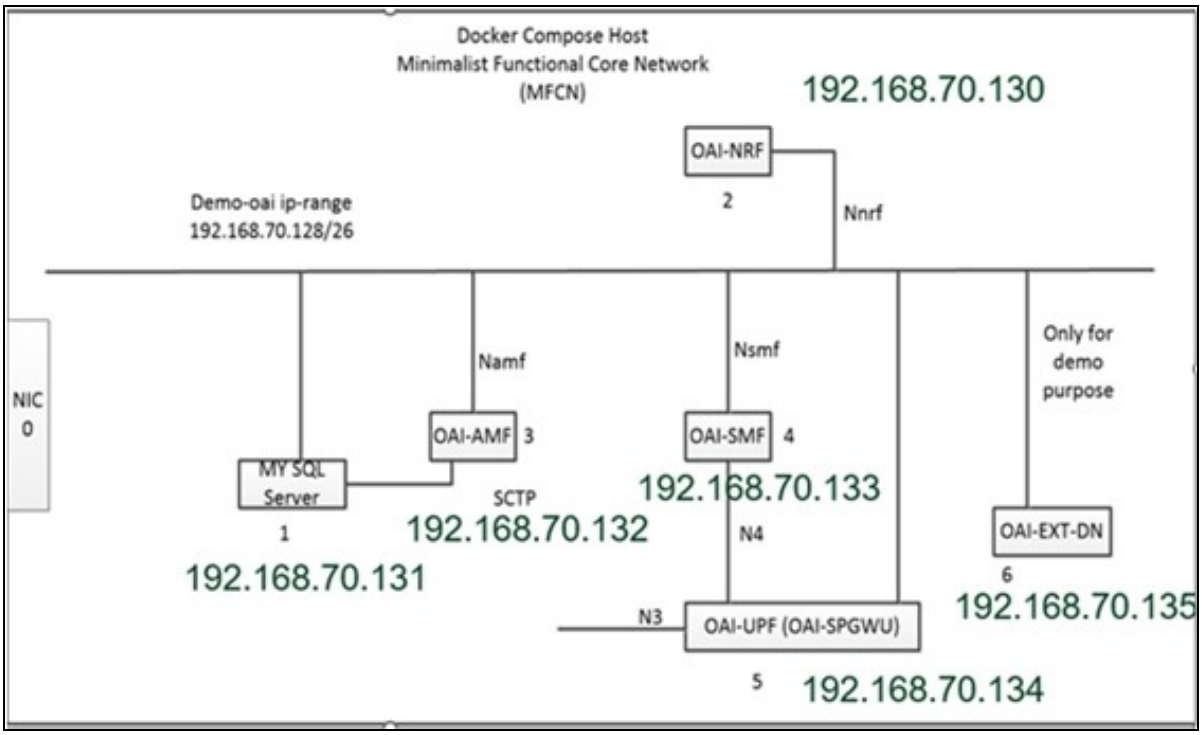
This Application Note explains only the minimalist deployment using docker-compose.

The minimalist deployment can be performed in two scenarios:

- Scenario 1: AMF, SMF, UPF (SPGWU), NRF, MYSQL
- Scenario 2: AMF, SMF, UPF (SPGWU), MYSQL

This Application Note only discusses Scenario 1.

The diagram below shows the layout and architecture of the various Docker containers running on the CN machine, along with the IP address of each container.



Verify that you are running docker-compose version 1.25 or higher.

```
dpkg --list | grep docker
```

Verify that you are running Python version 3.6 or higher.

```
python3 --version
```

Note that you should add your username to the `docker` group so that you can run Docker operations without needing root privileges.

To add your username to the `docker` group, use the command below.

```
sudo usermod -a -G docker yourusername
```

In order to pull the Docker images for the different network functions of the 5G CN, you need to have an account on docker-hub. Use this link to create an account.

We need to pull two base Docker images `ubuntu:bionic` and `mysql:5.7`.

Log in with your Docker Hub credentials, and push and pull images from Docker Hub.

```
docker login
docker pull ubuntu:bionic
docker pull mysql:5.7
docker logout
```

Be sure to set the following network configuration settings. You must run these commands every time you restart the CN machine, as they will not persist across a reboot.

```
sudo sysctl net.ipv4.conf.all.forwarding=1
sudo iptables -P FORWARD ACCEPT
```

The images are hosted under the OAI account `oaisoftwarealliance`.

You may need to login to Docker Hub.

```
docker login
```

Pull the images listed below.

```
docker pull rdefosseoai/oai-amf:latest
docker pull rdefosseoai/oai-nrf:latest
docker pull rdefosseoai/oai-spgwu-tiny:latest
docker pull rdefosseoai/oai-smf:latest
docker pull rdefosseoai/oai-udr:latest
docker pull rdefosseoai/oai-udm:latest
docker pull rdefosseoai/oai-ausf:latest
docker pull rdefosseoai/oai-upf-vpp:latest
docker pull rdefosseoai/oai-nssf:latest
```

Re-tag the images so that the docker-compose files for the tutorial work.

```
docker image tag rdefosseoai/oai-amf:latest oai-amf:latest
docker image tag rdefosseoai/oai-nrf:latest oai-nrf:latest
docker image tag rdefosseoai/oai-smf:latest oai-smf:latest
docker image tag rdefosseoai/oai-spgwu-tiny:latest oai-spgwu-tiny:latest
docker image tag rdefosseoai/oai-udr:latest oai-udr:latest
docker image tag rdefosseoai/oai-udm:latest oai-udm:latest
```

```
docker image tag rdefosseoai/oai-ausf:latest oai-ausf:latest
docker image tag rdefosseoai/oai-upf-vpp:latest oai-upf-vpp:latest
docker image tag rdefosseoai/oai-nssf:latest oai-nssf:latest
```

Finally, you may logoff. Your token is stored in plain-text.

```
docker logout
```

Clone the Git repository for the Core Network, and checkout the `v1.3.0` tag.

```
git clone --branch v1.3.0 https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed.git
```

You may also do this in two discrete steps.

```
git clone https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed.git
git checkout v1.3.0
```

Next, go into the top-level folder in the repository.

```
cd oai-cn5g-fed
```

Next, synchronize all the Git submodules.

```
./scripts/syncComponents.sh

------------------------------------------------------------
OAI-NRF     component branch : master
OAI-AMF     component branch : master
OAI-SMF     component branch : master
OAI-SPGW-U  component branch : master
OAI-AUSF    component branch : master
OAI-UDM     component branch : master
OAI-UDR     component branch : master
OAI-UPF-VPP component branch : master
OAI-NSSF    component branch : master
------------------------------------------------------------

git submodule deinit --all --force
git submodule init
git submodule update
```

Configure the Core Network by editing the `docker-compose-mini-nrf.yaml` file and specifying the proper PLMN, TAC, Operator Key, and DNN, according to the gNB and the UE being used.

If you are using the `v1.3.0` tag for the CN code, then specify the PLMN, TAC, and Operator Key in the `amf.conf` file, according to the gNB and the UE being used. If you are using the newer `v1.3.0` tag for the CN code, then do not do this.

User subscription information should be present in the MySQL database before trying to connect the UE. This can be done by adding the UE information into the `oai_db1.sql` file. Adjust the values shown below with your actual values.

```
INSERT INTO users VALUES
    (imsi,msisdn,imei,NULL,'PURGED',50,40000000,100000000,47,0000000000,1,key,0,0,0x40,'ebd07771ace8677a',opc);
```

The containers must be deployed in a strict order for AMF, SMF, and UPF registration with NRF. The data flow sequence must be from the MySQL database, to OAI-NRF, to OAI-AMF, to OAI-SMF, to OAI-UPF.

Install `docker-compose` onto the Core Network system. The link below provides detailed instructions.

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04

Install Wireshark with the commands listed below.

```
sudo add-apt-repository ppa:wireshark-dev/stable
sudo apt update
sudo apt install wireshark
```

Check the Wireshark version, and confirm that it is at least version 3.4.7.

```
wireshark --version
```

The Core Network is invoked using a Python script, which is a wrapper for docker-compose and other Docker commands. The script informs the user when the Core Network is correctly configured by checking the health status of containers and checking the connectivity between different Core Network components. Run the commands below.

```
cd oai-cn5g-fed/docker-compose
sudo  python3 ./core-network.py --type start-mini --fqdn no --scenario 1
```

After running the command, you should see the output listed below.

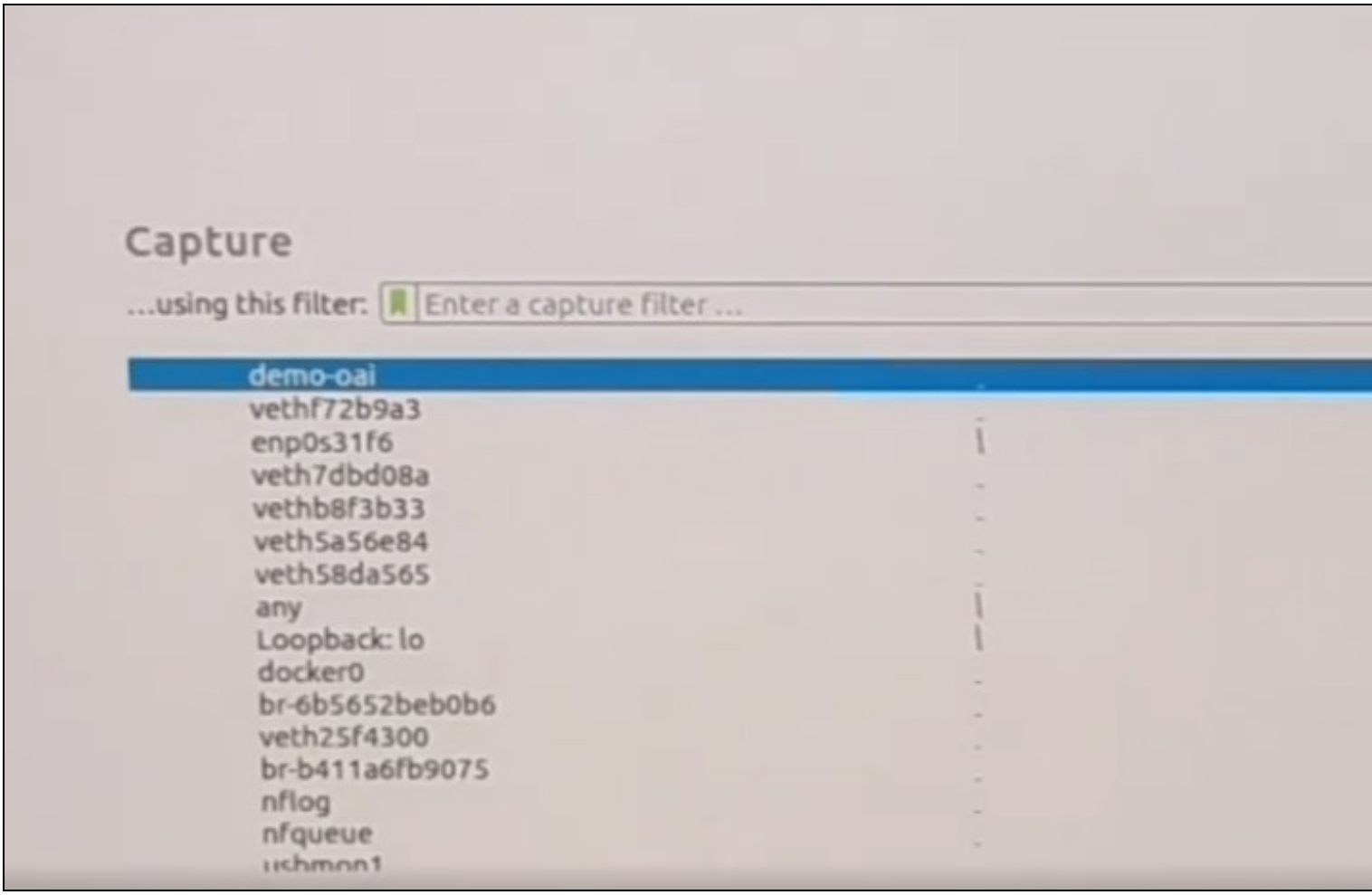
```
root:DEBUG:  Starting 5gcn components... Please wait....
Creating oai-nrf ... done
Creating mysql   ... done
Creating oai-amf ... done
Creating oai-smf ... done
Creating oai-spgwu ... done
Creating oai-ext-dn ... done
[2021-09-14 16:44:10,098] root:DEBUG:  OAI 5G Core network started, checking the health status of the containers... takes few secs....
[2021-09-14 16:44:47,025] root:DEBUG:  All components are healthy, please see below for more details....
Name                Command                  State                 Ports
-------------------------------------------------------------------------------------
mysql        docker-entrypoint.sh mysqld      Up (healthy)   3306/tcp, 33060/tcp
oai-amf      /bin/bash /openair-amf/bin ...   Up (healthy)   38412/sctp, 80/tcp, 9090/tcp
oai-ext-dn   /bin/bash -c  apt update;  ...   Up
```

```
oai-nrf      /bin/bash /openair-nrf/bin ...   Up (healthy)   80/tcp, 9090/tcp
oai-smf      /bin/bash /openair-smf/bin ...   Up (healthy)   80/tcp, 8805/udp, 9090/tcp
oai-spgwu    /openair-spgwu-tiny/bin/en ...   Up (healthy)   2152/udp, 8805/udp
[2021-09-14 16:44:47,025] root:DEBUG:  Checking if the containers are configured....
[2021-09-14 16:44:47,025] root:DEBUG:  Checking if SMF and UPF registered with nrf core network....
[2021-09-14 16:44:47,059] root:DEBUG:  For example: oai-smf Registration with oai-nrf can be checked on this url /nnrf-nfm/v1/nf-instances
[2021-09-14 16:44:47,059] root:DEBUG:  SMF and UPF are registered to NRF....
[2021-09-14 16:44:47,059] root:DEBUG:  Checking if SMF is able to connect with UPF....
[2021-09-14 16:44:47,176] root:DEBUG:  UPF receiving heathbeats from SMF....
[2021-09-14 16:44:47,176] root:DEBUG:  OAI 5G Core network is configured and healthy....
```

After launching the OAI 5G Core Network, run Wireshark from another terminal.

```
sudo wireshark
```

Once Wireshark has launched, you should see the `demo-oai` network interface listed. Open the `demo-oai` network interface, and confirm that all the packets exchanged between the different containers such as NRF, AMF, SMF, UPF are shown.



The `demo-oai` network interface can also be created manually. Since this is not the default behavior, you would need to edit the `docker-compose` file. The bottom section of the `docker-compose-mininrf.yaml` should be configured as shown below.

```
networks:
    public_net:
        external:
            name: demo-oai-public-net
    # public_net:
    #     driver: bridge
    #     name: demo-oai-public-net
    #     ipam:
    #         config:
    #             - subnet: 192.168.70.128/26
    #     driver_opts:
    #         com.docker.network.bridge.name: "demo-oai"
```

The `docker-compose-host` machine needs to be configured with the `demo-oai` network interface before deploying the core network components. This is needed in order to capture the initial message exchange between the SMF-to-NRF-to-UPF.

```
(docker-compose-host)v$ docker network create \
  --driver=bridge \
  --subnet=192.168.70.128/26 \
  -o "com.docker.network.bridge.name"="demo-oai" \
  demo-oai-public-net
455631b3749ccd6f10a366cd1c49d5a66cf976d176884252d5d88a1e54049bc5
(docker-compose-host)$ ifconfig demo-oai
demo-oai: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.70.129  netmask 255.255.255.192  broadcast 192.168.70.191
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
(docker-compose-host)$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
```

```
d2d34e05bb2d        bridge              bridge              local
455631b3749c        demo-oai-public-net bridge              local
```

We will next create a bridge automatically. The bridge can be automatically created using the `docker-compose` file, if there is no need to capture initial packets. This is the default version in the `docker-compose-mini-nrf.yaml` file. The bottom section should be configured as shown below.

```
networks:
    # public_net:
    #     external:
    #         name: demo-oai-public-net
      public_net:
          driver: bridge
          name: demo-oai-public-net
          ipam:
              config:
                  - subnet: 192.168.70.128/26
          driver_opts:
              com.docker.network.bridge.name: "demo-oai"
```

The screenshots listed below show the Wireshark packet captures for various components when the bridge is built manually.

Showing the NRF and AMF:

```
  3 2.959854473   192.168.70.132   192.168.70.130   TCP    74 57964 → 80 [SYN] Seq=0 W
  4 2.959876019   192.168.70.130   192.168.70.132   TCP    74 80 → 57964 [SYN, ACK] Se
```

Showing the NRF and SMF:

```
 13 4.176452769   192.168.70.133   192.168.70.130   TCP    74 59742 → 80 [SYN] Seq=0 W
 14 4.176473640   192.168.70.130   192.168.70.133   TCP    74 80 → 59742 [SYN, ACK] Se
```

Showing the SMF and UPF:

```
 33 5.958342193   192.168.70.134   192.168.70.133   PFCP   79 PFCP Association Setup R
 34 5.964381059   192.168.70.133   192.168.70.134   PFCP   77 PFCP Association Setup R
```

Showing the NRF and UPF:

```
 35 6.018142673   192.168.70.134   192.168.70.130   TCP    74 48226 → 80 [SYN] Seq=0 Wi
 36 6.018164174   192.168.70.130   192.168.70.134   TCP    74 80 → 48226 [SYN, ACK] Seq
```

Showing the SMF request to NRF for subscribing UPF registration/deregistration events:

```
 16 5.254114218   192.168.70.133   192.168.70.130   HTTP/JSON   444 POST /nnrf-nfm/v1/
```

Showing the SMF registration with NRF for the PUT request:

```
 26 5.266108102   192.168.70.133   192.168.70.130   HTTP/JSON   982 PUT /nnrf-nfm/v1/n
```

Showing the UPF registration with NRF for the PUT request:

```
 38 7.285212938   192.168.70.134   192.168.70.130   HTTP/JSON   596 PUT /nnrf-nfm/v1/n
```

Showing the NRF notification to SMF for UPF registration for the POST request:

```
 43 7.285636623   192.168.70.130   192.168.70.133   HTTP/JSON   707 POST /nsmf-nfstat
```

Showing the SMF-to-UPF PFCP association setup request-and-response:

```
 45 7.295097217   192.168.70.133   192.168.70.134   PFCP   72 PFCP Association Se
 46 7.295283861   192.168.70.134   192.168.70.133   PFCP   84 PFCP Association Se
```

These Wireshark packet captures are only visible if you create the `demo-oai` bridge manually. If the bridge is built automatically, then you will not observe these Wireshark packet captures.

In order to halt the operation of the Core Network, run the commands listed below.

```
cd oai-cn5g-fed/docker-compose
sudo python3 ./core-network.py --type stop-mini --fqdn no --scenario 1
```

Before installing and configuring the gNB system, be sure that you first have configured the system BIOS, implemented the Linux performance tuning settings, and installed UHD.

The commands listed below build and install the gNB software from source code using the OAI Git repository.

```
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
cd openairinterface5g
git checkout 2022.w33
source oaienv
cd cmake_targets
```

Confirm that you are using the correct branch and commit hash. Run the command listed below, and verify that the branch is `2022.w33` and that the commit hash is `ad8381a66bb67d6ef7fa94c9a6ae6c66f3ae84b8`.

```
git status
```

Edit the `build_oai` script, and comment out the lines shown below, by adding a hash mark `#` at the beginning of each line.

```
if [ "$HW" == "OAI_USRP" ] ; then
  echo_info "installing packages for USRP support"
  #check_install_usrp_uhd_driver
  #if [ ! "$DISABLE_HARDWARE_DEPENDENCY" == "True" ]; then
  #  install_usrp_uhd_driver $UHD_IMAGES_DIR
```

```
  #fi
fi
```

For the very first time that you build the gNB software, use the command listed below, which includes the "-I" option, which installs the package dependencies for the gNB software.

```
./build_oai -I --w USRP
```

For successive builds, use the command listed below, which omits the "-I" option.

```
./build_oai --gNB --w USRP
```

Once the gNB software is built and installed, there are specific settings that need to be made in the gNB configuration file before we can run the gNB software. The folder `openairinterface5g/ci_scripts/conf_files` contains various configuration files. For this Application Note, we will use the configuration file `band78.sa.fr1.106PRB.usrpn310.conf` . In this file, change the parameters in the PLMN section. Ensure that the PLMN configuration values are the same as the values used for the CN system, specifically `tracking_area_code` , `mcc` , `mnc` , `sst` , and `sd` .

```
gNB_name  =  "gNB-OAI";
min_rxtxtime_pdsch = 6;

// Tracking area code, 0x0000 and 0xfffe are reserved values
tracking_area_code  =  40960;

plmn_list = ({
              mcc = 208;
              mnc = 92;
              mnc_length = 2;
              snssaiList = (
                            {
                              sst = 1;
                              sd  = 0x00007b; // 0 false, else true
                            },
                            {
                              sst = 1;
                              sd  = 0x00000c; // 0 false, else true
                            }
                           );
            });

nr_cellid = 12345678L
```

The correct IP address for the AMF needs to be defined in the configuration file, as shown below. In the configuration file, in the `AMF parameters` section, in the `ipv4` field, add the IP address of the AMF Docker container. In this reference architecture, it is 192.168.70.132.

```
////////// AMF parameters:
    amf_ip_address      = ( { ipv4       = "192.168.70.132";
                              ipv6       = "192:168:30::17";
                              active     = "yes";
                              preference = "ipv4";
```

The correct IP address for the gNB needs to be defined in the configuration file, as shown below. In the configuration file, in the `NETWORK_INTERFACES` section, add the IP address of the gNB system to the `GNB_IPV4_ADDRESS_FOR_NG_AMF` and the `GNB_IPV4_ADDRESS_FOR_NGU` fields. In this reference architecture, it is 10.89.14.37.

```
NETWORK_INTERFACES :
{

    GNB_INTERFACE_NAME_FOR_NG_AMF            = "demo-oai";
    GNB_IPV4_ADDRESS_FOR_NG_AMF              = "10.89.14.37";
    GNB_INTERFACE_NAME_FOR_NGU               = "demo-oai";
    GNB_IPV4_ADDRESS_FOR_NGU                 = "10.89.14.37";
    GNB_PORT_FOR_S1U                         = 2152; # Spec 2152
};
```

The correct IP address for the USRP needs to be defined in the configuration file, as shown below. In the configuration file, in the `RU` section, in the `sdr_addrs` field, we need to specify the device arguments for the USRP, which includes the type of the USRP (`type`), the mangement IP address (`mgmt_addr`), the primary streaming IP address (`addr`), the secondary streaming IP address (`second_addr`), the clocking source (`clock_source`), and the timing source (`time_source`). In this reference architecture, we use the device argument string listed below.

```
sdr_addrs="type=x4xx,mgmt_addr=192.168.10.2,addr=192.168.10.2,second_addr=192.168.20.2,clock_source=external,time_source=external"
```

The `mgmt_addr`, `addr`, and `second_addr` should be defined according to the configuration of your USRP device. For the `clock_source` and the `time source`, the values can be `internal`, `external`, or `gpsdo`. In this reference architecture, an OctoClock-G is connected to the USRP, so we use the value of `external` for both the clocking source and the timing source.

To enable communication between the gNB and CN machine, a static route must be added to the gNB machine. Note that this added route is not permanent, and when the system is restarted, the static route will need to be added again. The command listed below will add the static route.

```
sudo ip route add 192.168.70.128/26 via 10.89.14.119 dev eno1
```

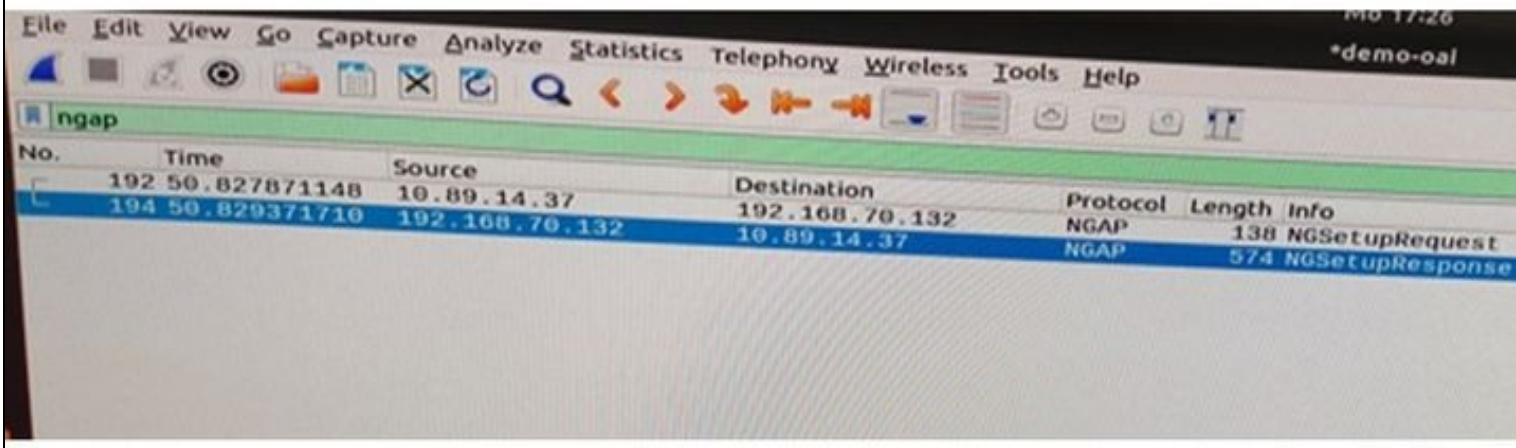
In this command, the IP address 192.168.70.128/26 is for the `demo-oai` network bridge created when the CN is invoked, and the IP address 10.89.14.119 is for the CN machine, and the `eno1` Ethernet interface is the port on the gNB machine connected to the CN machine.

On your machine, the Ethernet interface name will likely be different, but you can use the same IP address for the CN machine, and the same static IP address for the CN machine, as used in this reference architecture.

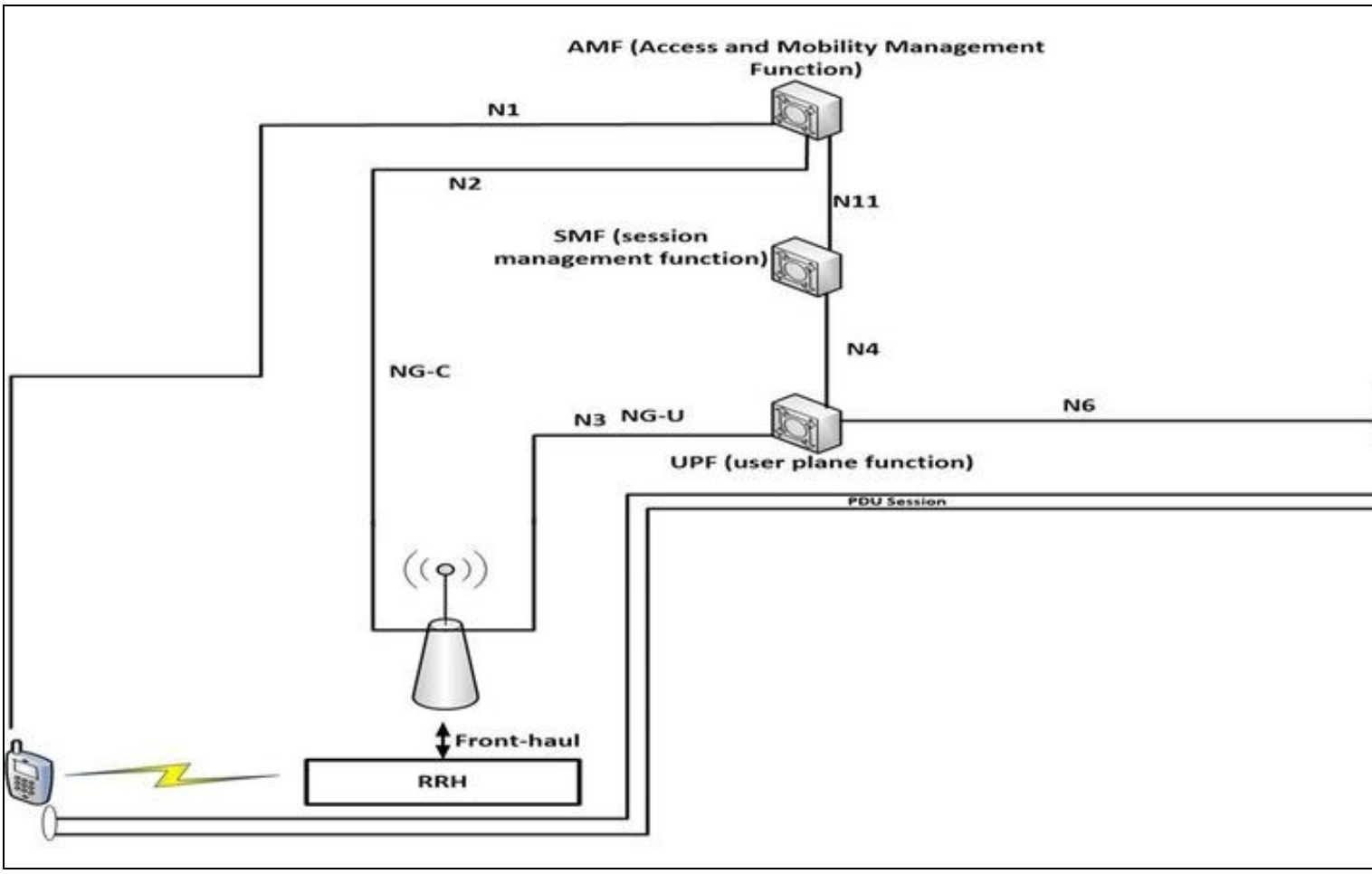
In order to invoke the gNB, we first need to copy the edited gNB configuration file from the folder `openairinterface5g/ci_scripts/conf_files` to the folder `openairinterface5g/targets/PROJECTS/GENERIC-NR-5G/CONF`. Then, the command listed below will invoke the gNB.

```
sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/gnb.band78.sa.fr1.106PRB.usrpn310.conf --sa --usrp-tx-thread-config 1
```

Launch Wireshark, select the Ethernet interface connected to the CN machine, and watch for NGAP packets. As soon as the gNB is launched, the CN and gNB will exchange NGAP setup request and response messages. In these messages, the gNB and CN check the MCC, MNC and TAC parameters. If these values are identical on both the gNB machine and the CN machine, then the NGAP setup request and response will be successful.



When the gNB is invoked, the N2 and N3 interfaces will be created. The N2 interface supports the NGAP protocol, and the N3 interface supports the GTP protocol. The N1 interface is created when UE attaches to the network and supports the NAS protocol.



Once both the gNB and the CN machines are up-and-running, test the connectivity between the two systems by pinging the AMF in the CN machine from the gNB machine. On the gNB machine, run the command listed below.

```
ping 192.168.70.132
```

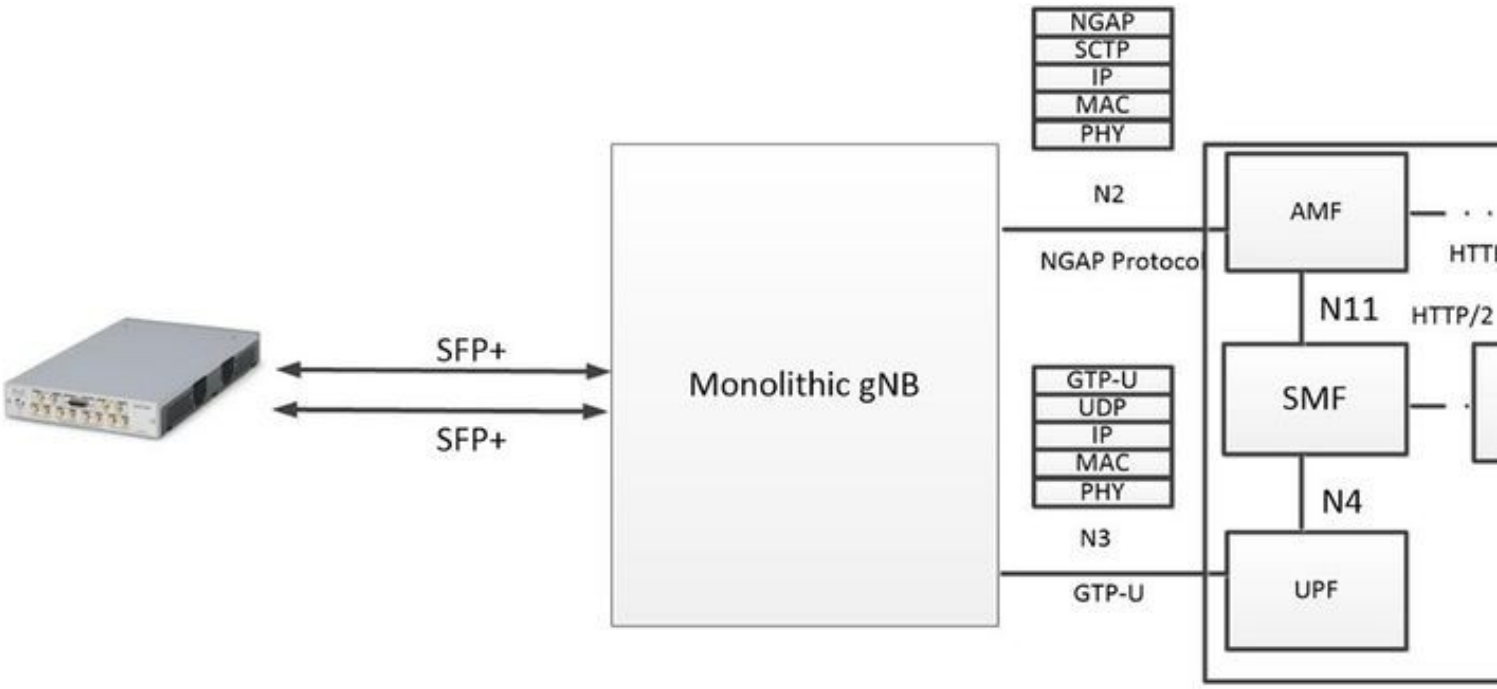
If the ping fails, then it may be due to the firewall settings. On the CN machine, run the commands listed below to configure the firewall.

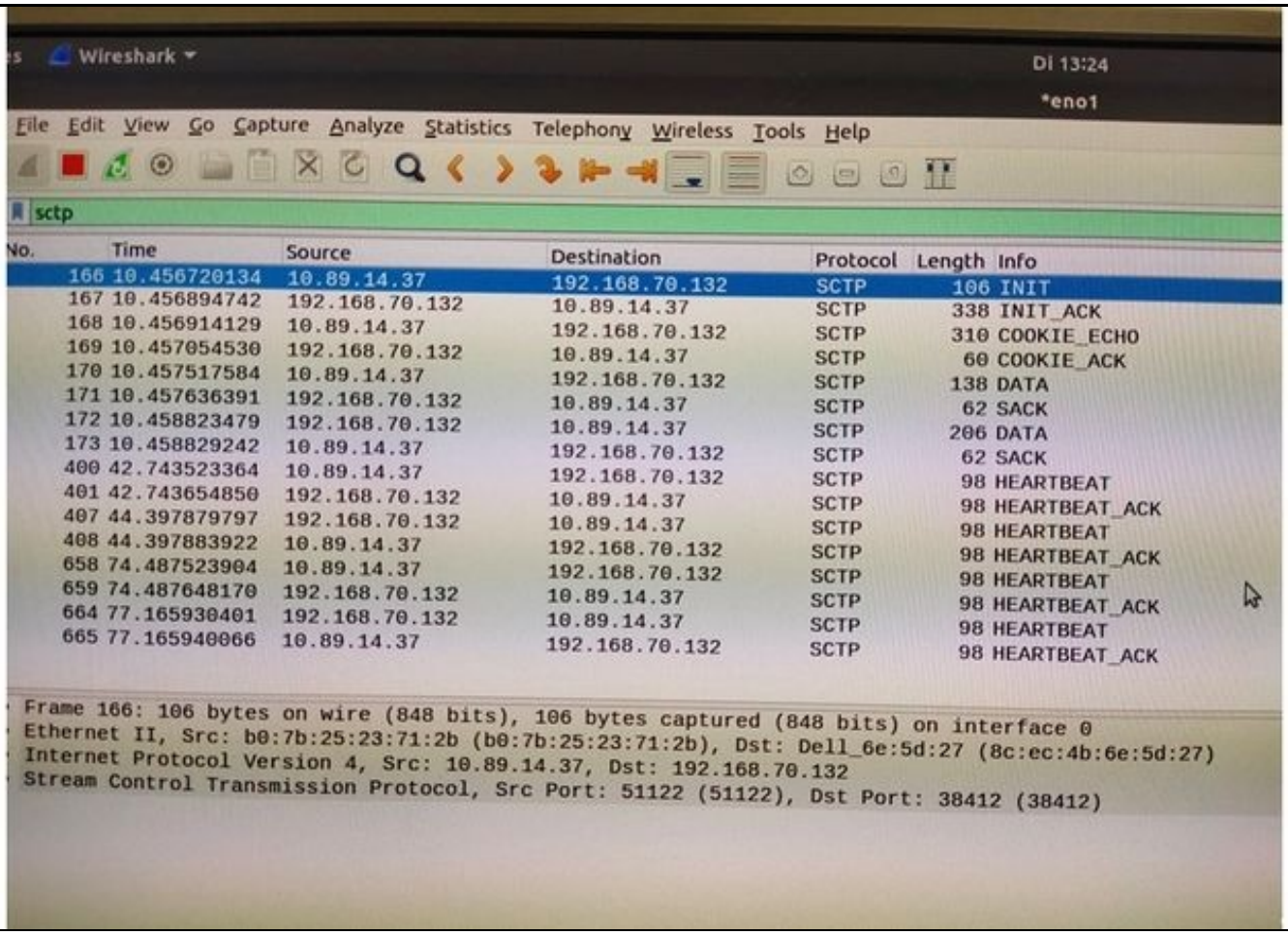
```
sysctl net.ipv4.conf.all.forwarding=1
sudo iptables -P FORWARD ACCEPT
```

After running these commands on the CN machine, run this command on the gNB machine to add a route from the gNB machine to the CN machine. You will likely need to replace the Ethernet interface name `eno1` with the correct interface name on your gNB machine.

```
sudo ip route add 192.168.70.128/26 via 10.89.14.6 dev eno1
```

The IP address 192.168.70.128/26 is for the `demo-oai` interface, and 10.89.14.6 is the IP address of the CN machine, and `eno1` is the Ethernet interface name on the gNB machine.


NGAP
SCTP
IP
MAC
PHY
N2
NGAP Protocol
AMF
HTTP
N11
HTTP/2
SFP+
SFP+
Monolithic gNB
GTP-U
UDP
IP
MAC
PHY
N3
GTP-U
SMF
N4
UPF


Wireshark
Di 13:24
*eno1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
sctp
No. Time Source Destination Protocol Length Info
166 10.456720134 10.89.14.37 192.168.70.132 SCTP 106 INIT
167 10.456894742 192.168.70.132 10.89.14.37 SCTP 338 INIT_ACK
168 10.456914129 10.89.14.37 192.168.70.132 SCTP 310 COOKIE_ECHO
169 10.457054530 192.168.70.132 10.89.14.37 SCTP 60 COOKIE_ACK
170 10.457517584 10.89.14.37 192.168.70.132 SCTP 138 DATA
171 10.457636391 192.168.70.132 10.89.14.37 SCTP 62 SACK
172 10.458823479 192.168.70.132 10.89.14.37 SCTP 206 DATA
173 10.458829242 10.89.14.37 192.168.70.132 SCTP 62 SACK
400 42.743523364 10.89.14.37 192.168.70.132 SCTP 98 HEARTBEAT
401 42.743654850 192.168.70.132 10.89.14.37 SCTP 98 HEARTBEAT_ACK
407 44.397879797 192.168.70.132 10.89.14.37 SCTP 98 HEARTBEAT
408 44.397883922 10.89.14.37 192.168.70.132 SCTP 98 HEARTBEAT_ACK
658 74.487523904 10.89.14.37 192.168.70.132 SCTP 98 HEARTBEAT
659 74.487648170 192.168.70.132 10.89.14.37 SCTP 98 HEARTBEAT_ACK
664 77.165930401 192.168.70.132 10.89.14.37 SCTP 98 HEARTBEAT
665 77.165940066 10.89.14.37 192.168.70.132 SCTP 98 HEARTBEAT_ACK
Frame 166: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface 0
Ethernet II, Src: b0:7b:25:23:71:2b (b0:7b:25:23:71:2b), Dst: Dell_6e:5d:27 (8c:ec:4b:6e:5d:27)
Internet Protocol Version 4, Src: 10.89.14.37, Dst: 192.168.70.132
Stream Control Transmission Protocol, Src Port: 51122 (51122), Dst Port: 38412 (38412)

There are three scenarios for the UE implementation: the USRP radio; the 5G wireless modem module; and the COTS handset.

The UE can be implemented by running the OAI softmodem with a USRP N300, N310, N320, N321, or X410 device. Currently, this reference architecture is focused on the use of a USRP X410 device, although the integration of the other USRP devices is similar to the integration of the USRP X410 and is relatively straight-forward.

Before installing and configuring the UE system, be sure that you first have configured the system BIOS, implemented the Linux performance tuning settings, and installed UHD.

Note that the procedure for building and configuring the UE software is very similar to the procedure for building and installing the gNB software.

The commands listed below build and install the UE software from source code using the OAI Git repository.

```
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
cd openairinterface5g
git checkout 2022.w33
source oaienv
cd cmake_targets
```

Confirm that you are using the correct branch and commit hash. Run the command listed below, and verify that the branch is `2022.w33` and that the commit hash is `ad8381a66bb67d6ef7fa94c9a6ae6c66f3ae84b8`.

```
git status
```

Edit the `build_oai` script, and comment out the lines shown below, by adding a hash mark `#` at the beginning of each line.

```
if [ "$HW" == "OAI_USRP" ] ; then
  echo_info "installing packages for USRP support"
  #check_install_usrp_uhd_driver
  #if [ ! "$DISABLE_HARDWARE_DEPENDENCY" == "True" ]; then
  #  install_usrp_uhd_driver $UHD_IMAGES_DIR
  #fi
fi
```

For the very first time that you build the gNB software, use the command listed below, which includes the "-I" option, which installs the package dependencies for the gNB software.

```
./build_oai -I --w USRP
```

For successive builds, use the command listed below, which omits the "-I" option.

```
./build_oai --gNB --w USRP
```

Once we build and install the UE software, we need to modify the UE configuration file. Only one UE configuration file is available in the folder `openairinterface5g/targets/PROJECTS/GENERIC-NR-5GC/CONF`. The UE configuration file used in this reference architecture is `ue.conf`. The default contents of the file is shown below.

```
uicc0 = {
imsi = "2089900007487";
key = "fec86ba6eb707ed08905757b1bb44b8f";
opc= "C42449363BBAD02B66D16BC975D77CC1";
dnn= "oai";
nssai_sst=1;
nssai_sd=1;
}
```

Edit the fields `imsi`, `key`, `opc`, `dnn`, `nssai_sst`, `nssai_sd` per the values for the CN. Earlier, when configuring the Docker containers for the CN system, we entered several parameters in the `oai_db1.sql` file. The `imsi`, `key`, `opc` fields there should match the values here in the UE configuration file. The `dnn` value should be same as listed in the `docker-compose-mini-nrf.yaml<code> file. The <code>nssai_sst` and `nssai_sd` values should be same as listed in the gNB configuration file and the `docker-compose-mini-nrf.yaml` file.

Run the command listed below to invoke the OAI UE softmodem.

```
openairinterface5g/cmake_targets/ran_build/build sudo ./nr-uesoftmodem --usrp-args "addr=192.168.10.2, clock_source=external,time_source=e
```

The command-line option `--usrp-args` specifies the USRP device arguments, which are passed to the UHD driver. Here we specify the IP address of the USRP, along with clocking source and the timing source, which are set to `external`, to indicate the use of the connection to the OctoClock-G device.

The command-line option `-r 106` specifies the PRBS being used. Based on the PRBS, we can calculate the operational channel bandwidth being used (i.e., 40 MHz).

The command-line option `--numerology 1` specifies the SCS being used (i.e., 30 KHz).

The command-line option `--band 78` specifies the 3GPP frequency band being used.

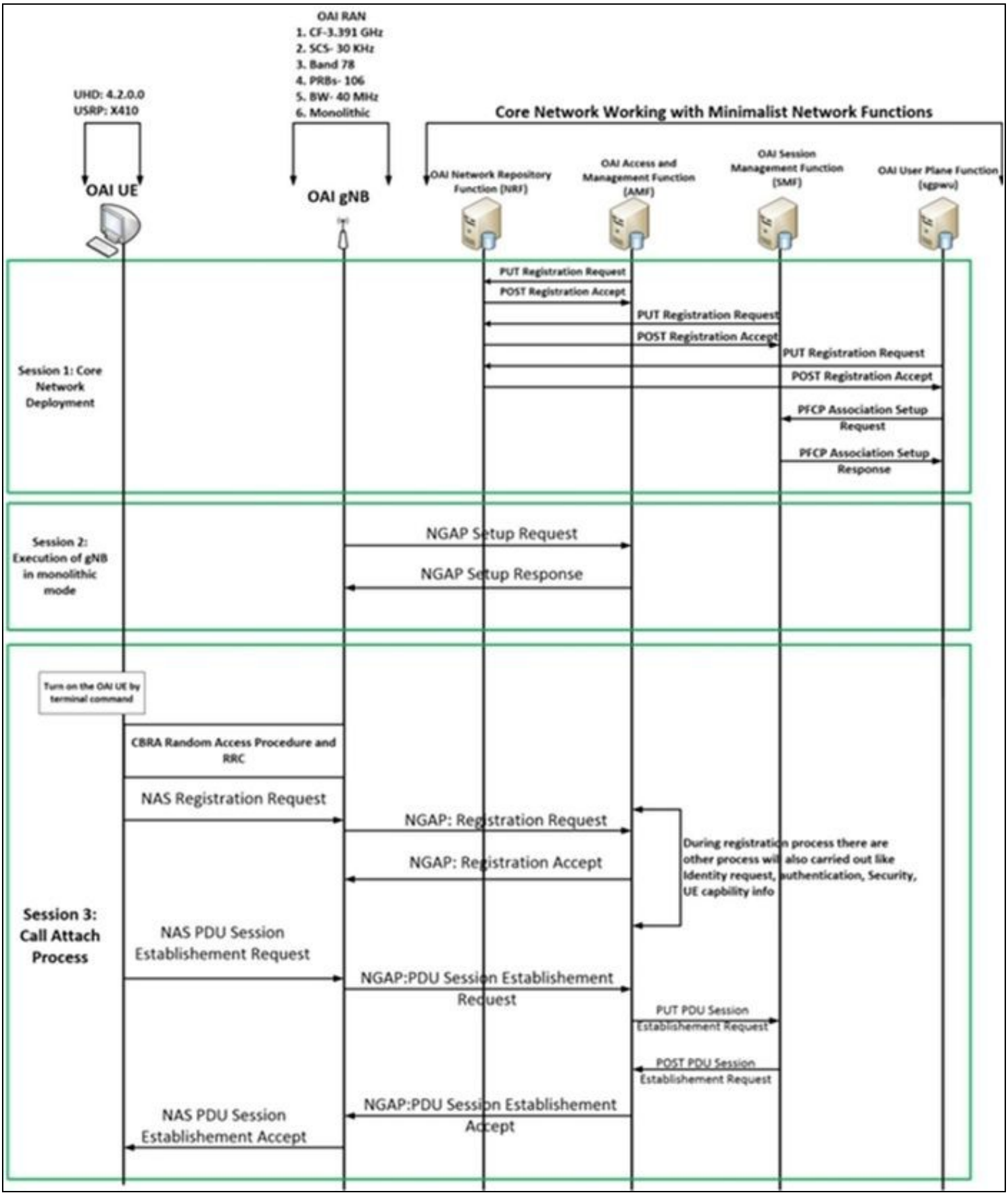
The command-line option `-C 3319680000` specifies the CF we are operating (i.e., 3.319 GHz).

The command-line option `--nokrnmod 1` forces the use of the tunnel interface.

Once the UE is attached to the CN, verify that the `oaitun_ue1` network interface has been created, and that is has been assigned an IP address of 10.x.x.x, using the command listed below.

```
ifconfig
```

The call attach process between the UE and the gNB is illustrated in the sequence diagram shown below.



The relevant lines in the OAI UE softmodem log file show information about the call attach process.

```
[PHY]    [SCHED][UE] Check absolute frequency DL 3319680000.000000, UL 3319680000.000000 (RF card 0, oai_exmt 0, channel 0, rx_num_channels 1)
[PHY]    Starting sync detection
[PHY]    [UE thread Synch] Running Initial Synch (mode 4)
[PHY]    [UE] nr_synchro_time: Sync source = 0, Peak found at pos 283468, val = 10497327265 (99 dB) avg 45 dB, ffo 0.000000
PSS execution duration 285041 microseconds
[PHY]    [UE0] Initial sync : Estimated PSS position 283468, Nid2 0
[PHY]    sync_pos 283468 ssb_offset 283324
[PHY]    Calling sss detection (normal CP)
Nid2 0 Nid1 0 tot_metric 41290, phase_max 2
[PHY]    [UE0] Initial sync: starting PBCH detection (rx_offset 0)
[NR_RRC]    Configuring MAC for first MIB reception
[PHY]    [UE0] Initial sync: pbch decoded sucessfully
[PHY]    TDD Normal prefix: CellId 0 metric 41290, phase 2, pbch 0
[PHY]    [UE0] In synch, rx_offset 278908 samples
[PHY]    [UE 0] RRC Measurements => rssi -inf dBm (dig -inf dB, gain 110), N0 0 dBm,  rsrp -inf dBm/RE, rsrq -inf dB
[PHY]    [UE 0] Measured Carrier Frequency 3319680000 Hz (offset 0 Hz)
[PHY]    HW: Configuring channel 0 (rf_chain 0): setting tx_freq 3319680000.000000 Hz, rx_freq 3319680000.000000 Hz
[PHY]    Got synch: hw_slot_offset 18, carrier off 0 Hz, rxgain 110.000000 (DL 3319680000.000000 Hz, UL 3319680000.000000 Hz)
Setting USRP TX Freq 3319680000.000000, RX Freq 3319680000.000000
[NR_RRC]    SIB1 decoded
[NR_MAC]    NR band duplex spacing is 0 KHz (nr_bandtable[37].band = 78)
[NR_MAC]    NR band 78, duplex mode TDD, duplex spacing = 0 KHz
[MAC]    Setting TDD configuration period to 6
[PHY]    TDD has been properly configurated
[MAC]    Initialising ul_config_request. num_slots_ul = 3
[NR_RRC]    [UE 0] : Logical Channel UL-CCCH (SRB0), Generating RRCSetupRequest (bytes 6, gNB 0)
[PHY]    Resynchronizing RX by 278908 samples (mode = 6)
[PHY]    !!!adjusting +1 samples!!! rx_offset == -4
[MAC]    Initialization of 4-step contention-based random access procedure
[PHY]    PRACH [UE 0] in slot 19, placing PRACH in position 2520, msg1 frequency start 0 (k1 0), preamble_offset 0, first_nonzero_root_idx 0
[NR_MAC]    [UE 0][RAPROC] Got SI RAR subPDU 5 ms
[NR_MAC]    [UE 0][RAPROC] Got RAPID RAR subPDU
[NR_MAC]    [UE 0][RAPROC][208.7] Found RAR with the intended RAPID 0
[NR_MAC]    [RAPROC][208.17] RA-Msg3 transmitted
[PHY]    !!!adjusting +1 samples!!! rx_offset == -4
[MAC]    [UE 0][RAPROC] Frame 210 : received contention resolution identity: 0x1ab77c0b3f04 Terminating RA procedure
[MAC]    [UE 0][210.1][RAPROC] RA procedure succeeded. CB-RA: Contention Resolution is successful.
```

Launch Wireshark to view the packets exchanged between the UE and gNB for this call attach process.

91	7.610958913	10.89.14.37	192.168.70.132	NGAP	126 NGSetupRequest
93	7.612847252	192.168.70.132	10.89.14.37	NGAP	574 NGSetupResponse
464	36.256949373	10.89.14.37	192.168.70.132	NGAP	126 NGSetupRequest
466	36.258622435	192.168.70.132	10.89.14.37	NGAP	574 NGSetupResponse
846	89.060510902	10.89.14.37	192.168.70.132	NGAP/NAS-5GS	146 InitialUEMessage, Registration request
847	89.068282411	192.168.70.132	10.89.14.37	NGAP/NAS-5GS	630 DownlinkNASTransport, Authentication request
848	89.219909741	10.89.14.37	192.168.70.132	NGAP/NAS-5GS	146 UplinkNASTransport, Authentication response
849	89.221168040	192.168.70.132	10.89.14.37	NGAP/NAS-5GS	462 DownlinkNASTransport, Security mode command
850	89.300181978	10.89.14.37	192.168.70.132	NGAP/NAS-5GS/NAS-5GS	174 UplinkNASTransport, Security mode complete, Registration request
851	89.302116707	192.168.70.132	10.89.14.37	NGAP/NAS-5GS	1390 InitialContextSetupRequest, Registration accept
852	89.370549572	10.89.14.37	192.168.70.132	NGAP	122 UERadioCapabilityInfoIndication
855	89.574529982	10.89.14.37	192.168.70.132	NGAP	86 InitialContextSetupResponse
861	90.400109630	10.89.14.37	192.168.70.132	NGAP/NAS-5GS	118 UplinkNASTransport, Registration complete
864	90.602530712	10.89.14.37	192.168.70.132	NGAP/NAS-5GS	146 UplinkNASTransport, UL NAS transport, PDU session establishment reque
865	90.607182312	192.168.70.132	10.89.14.37	NGAP/NAS-5GS	266 PDUSessionResourceSetupRequest, DL NAS transport, PDU session establi
866	90.700380878	10.89.14.37	192.168.70.132	NGAP	122 PDUSessionResourceSetupResponse

To verify that UE has successfully attached to the gNB, check the OAI-AMF logs on the CN machine using the command listed below. Check the output to confirm that UE has been registered by the Core Network.

```
sudo docker logs oai-amf
```

Run `ifconfig` on the UE machine to check the IP address assigned to the UE. Verify that the `oaitun_ue1` network interface has been created, and that is has been assigned an IP address of 10.x.x.x. You can also check the OAI-SMF logs using the command listed below.

```
sudo docker oai-smf logs
```

The UE can be implemented using a 5G wireless modem module. Currently, this reference architecture is focused on the use of a Quectel RM500Q-GL 5G wireless modem module. This device is a 5G NR sub-6GHz M.2 module which meets the 3GPP Release 15 specification, and is optimized for industrial and commercial IoT and eMBB applications. It supports both standalone (SA) and non-standalone (NSA) modes. Other wireless modem modules can certainly be used as well. Detailed documentation for the Sierra Wireless EM9191 module will be added in the near future.

When using a 5G wireless modem module or a COTS handset, a SIM card will be required. If a USRP is being used as the UE, running the OAI UE softmodem, then a SIM card is not required.

The SIM card used in this reference architecture is provided by Open-Cells, and is shown below. Note that the ADM code is printed directly on the SIM card itself.


open-cells.com
OC003302
OC003302

Insert the nano SIM card into the SIM card reder/writer, and plug it into the USB slot on the UE computer.

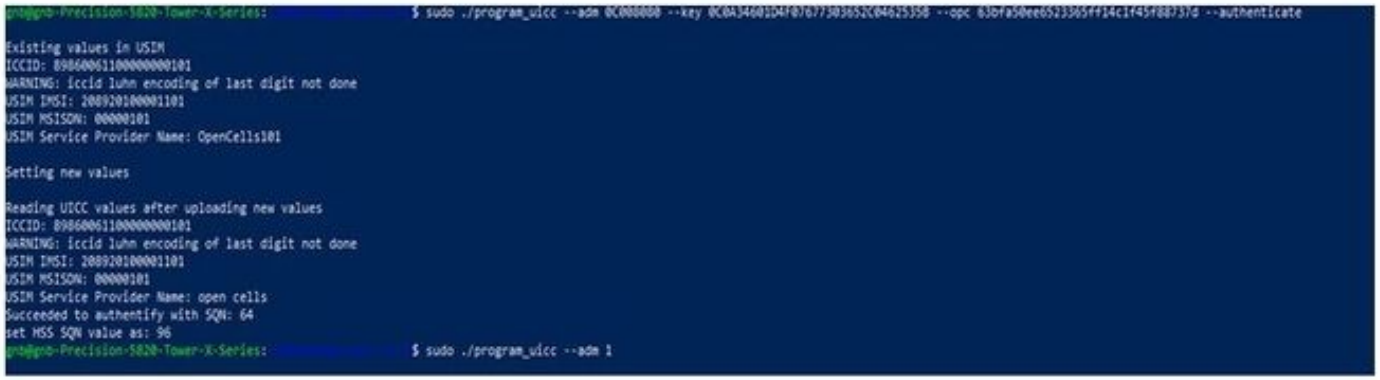
To read and program the SIM card, we use the program `program_uicc` from Open-Cells (here).

We first read the existing data on the SIM by running the command below.

```
sudo ./program_uicc --adm 1
```

We then write the key and the OPC in the UICC file in the SIM card. The ADM value enables this. Run the command below to perform this operation, where `ADM_VALUE_FROM_SIM` is the ADM value printed directly on the SIM card itself.

```
sudo ./program_uicc --adm <ADM_VALUE_FROM_SIM> --key 0C0A34601D4F07677303652C0462535B --opc 63bfa50ee6523365ff14c1f45f88737d --authenticat
```



Ensure that the values being programmed into the SIM card match the corresponding values entered in the SQL database on the CN machine. The values of primary importance are listed in the table below.

Primary Configuration Parameters for UE, gNB, CN

Parameter	UE	gNB	CN
IMSI	`208920100001101`	MCC: `208`, MNC: `92`	`208920100001101`
MSISDN	`00000101`		`00000101`
IMEI	`863305040549338`		`863305040549338`
Key	`0C0A34601D4F07677303652C0462535B`		`0C0A34601D4F07677303652C0462535B`
OPC	`63bfa50ee6523365ff14c1f45f88737d`		`63bfa50ee6523365ff14c1f45f88737d`

Attach all four antennas to the Quectel wireless modem module. Then, mount the Quectel module into the M.2 connector slot on the carrier board. Then, connect the carrier board to the UE computer via a USB 3.0 port.

We will use Minicom to communicate with the Quectel module over a USB serial connection. Run `which minicom` to verify that Minicom is already installed. If not, then run the command listed below to install it.

```
sudo apt-get install minicom
```

Once the Quectel module is plugged in, the Linux operating system should create several USB serial devices which can be used to communicate with the module. The default device should be `/dev/ttyUSB0`. Run the command listed below to start a Minicom serial console session with the Quectel device.

```
sudo minicom /dev/ttyUSB0
```

Note that in order to exit Minicom, type Ctrl-A, then "X".

We use Minicom to issue AT Commands to the 5G modem module.

There are informative articles about AT Commands here and here.

There are some specific AT Commands that we need to control and query the Quectel module. The AT Command are generally not case-sensitive. Execute all the AT Commands listed below, in-order, to setup and configure the Quectel module.

`AT+GMR`
: Display the current firmwave version number.

`AT+CIMI`
: Display the IMSI of the (U)SIM.

`AT+GSN`
: Display the IMEI of the (U)SIM.

`AT+QMBNCFG="select","ROW_Commercial"`
: Unlock the Quectel module.

`AT+qnwprefcfg="nr5g_band"`
: Display which 5G NR frequency bands are configured.

`AT+qnwprefcfg="mode_pref"`

Display which 5G NR mode is set.

`AT+qnwprefcfg="mode_pref",nr5g`
Set the operational mode to 5G NR SA.

`AT+qnwprefcfg="nr5g_disable_mode",0`
Enable 5G NR operations.

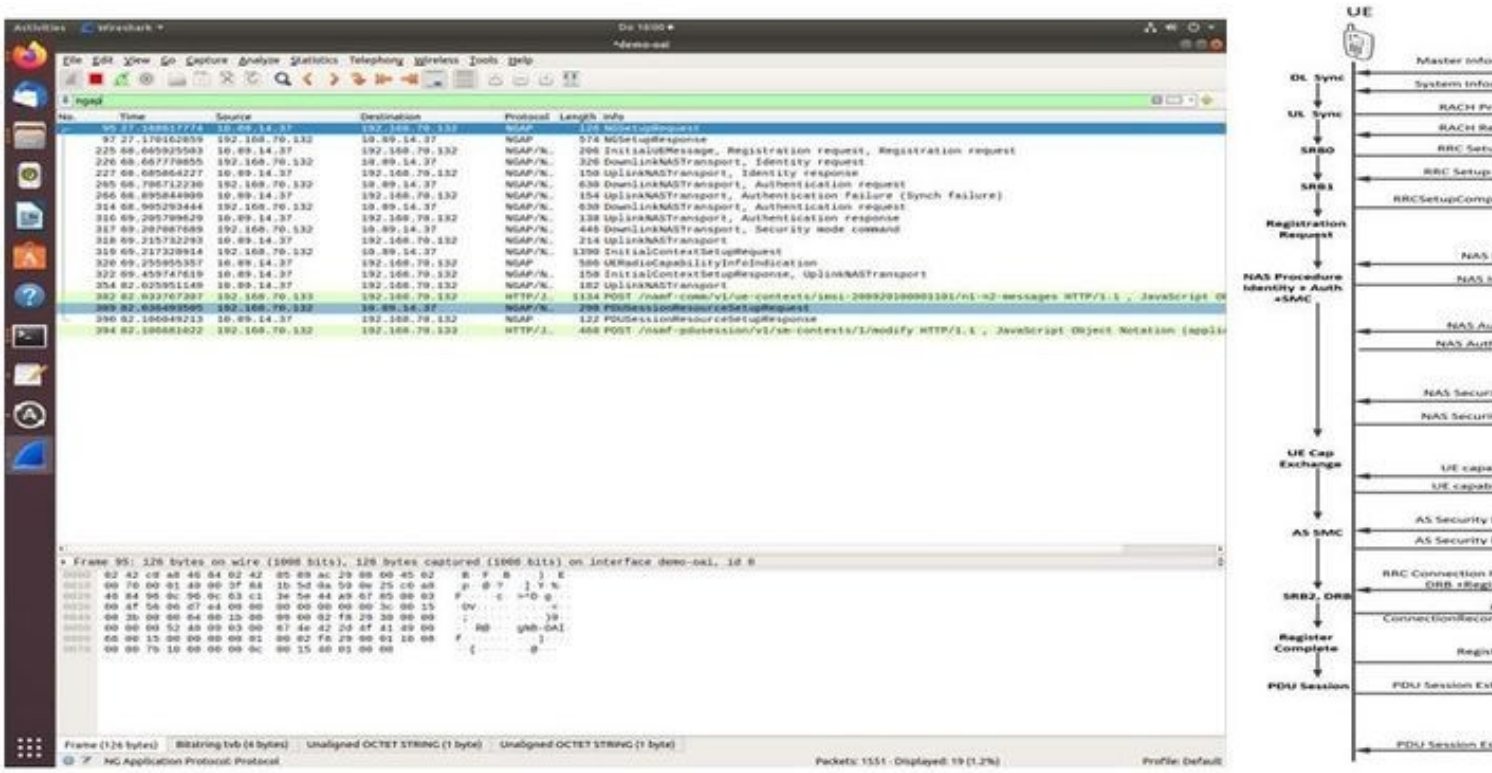
`AT+cgdcont=1,?IP?,?oai?,?0.0.0.0?,0,0 (cid, pdp type, apn, pdp_addr, data_comp, head-comp)`
Specify the PDP context parameters for a specific context cid.

`AT+CFUN=0`
Set the minimum functionality of the module.

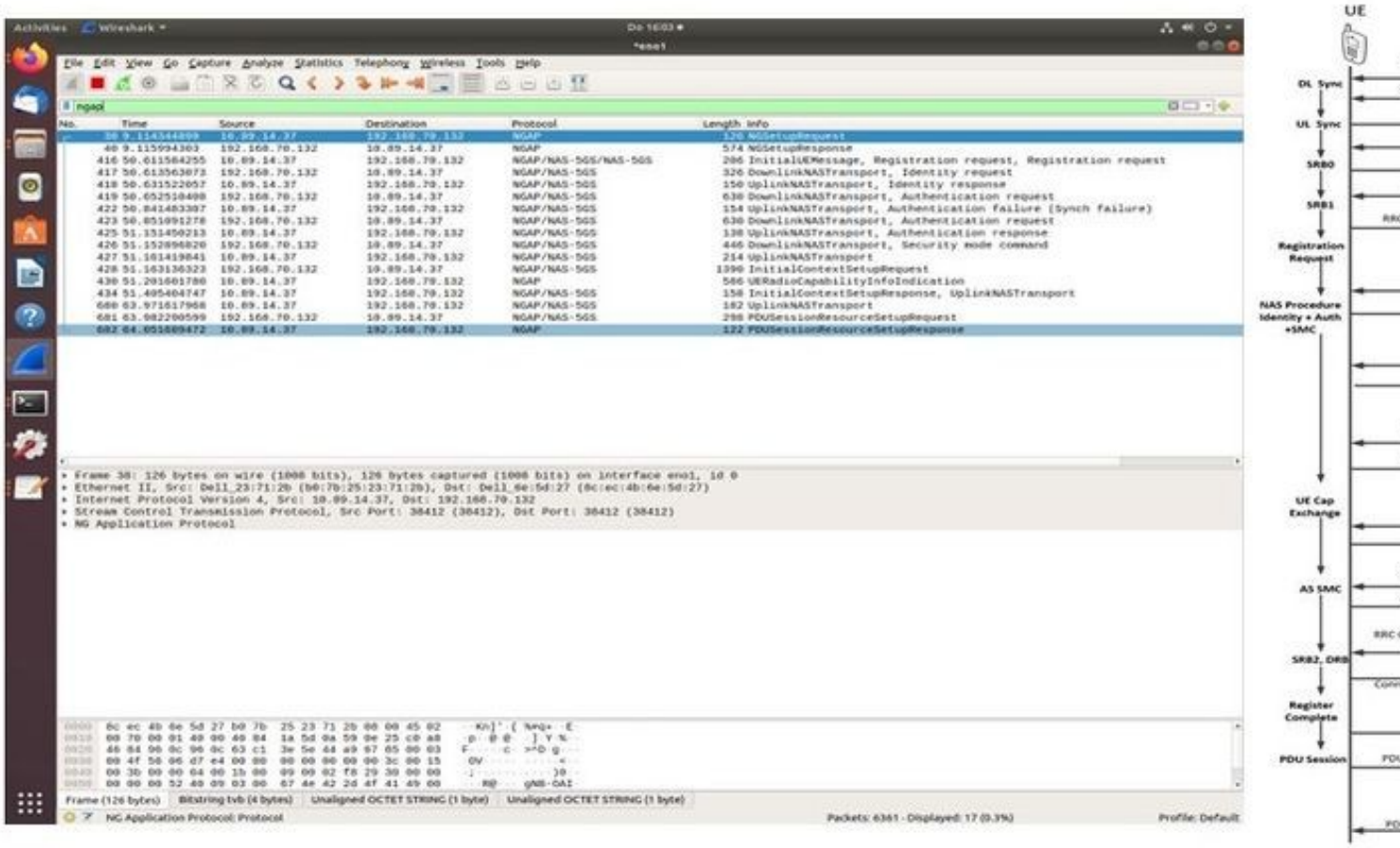
`AT+CFUN=1`
Set the full functionality of the module.

Once all the AT Commands have been executed, we should see msg1, msg2, msg3, msg4, RRC logs on the gNB and the registration request from the UE to the AMF on Wireshark.

The screenshot below shows Wireshark running on the CN machine, and shown on the right is the corresponding sequence diagram showing the message exchanges between the UE, gNB, and CN AMF, from the perspective of the CN machine.



The screenshot below shows Wireshark running on the gNB machine, and shown on the right is the corresponding sequence diagram showing the message exchanges between the UE, gNB, and CN AMF, from the perspective of the gNB machine.



We run the AT Commands listed below in Minicom, and examine the responses, to verify the operation of the system.

`AT+COPS?`

This output specifies the current operators and their status.
Expected Output: `0,0,"208 92 open cells",11`
Field 1: Availability of Operators, where `0` specifies Unknown
Field 2: Operator Selection of operators, where `0` specifies Automatic
Field 3: Operator Name, where `208 92 Open cell` specifies pre-programmed operator name
Field 4: Access Technology Selected, where `11` specifies 5G NR connected to a 5G CN

`AT+C5GREG?`

The output of this command shows the 5G network registration status.
Expected Output: `2,1,"1","0",11,16,"01.00007B;00.000000:01.00000C;00.000000"`
Field 1: Enable network registration and location information, where `2` specifies Unsolicited Mode
Field 2: Registration Status, where `1` specifies Registered on Home Network
Field 3: `1` specifies a three-byte tracking area code in hexadecimal format
Field 4: `0` specifies a five-byte (NR) cell ID in hexadecimal format
Field 5: `11` specifies that the connection mode is 5G NR connected to a 5G CN
Field 6: `16` specifies the number of octets of the allowed NSSAI information element
Field 7: `01.00007B;00.000000:01.00000C;00.000000` specifies the allowed NSSAI

The UE can be implemented using the commercial (COTS) handset. This reference architecture will feature the use of the Google Pixel 5A handset. Detailed documentation about this will be added in the near future.

We run `ping` on the CN machine to verify connectivity from the CN, through the gNB, to the UE. Run the command listed below on the CN machine. Replace `12.1.1.x` with the correct IP address of the UE, as assigned by the CN.

```
sudo docker exec ?it oai-ext-dn ping 12.1.1.x
```

We use `iperf` to measure the throughput in bits per second for the Downlink (DL) and for the Uplink (UL).

For the Downlink, the client runs on the CN machine, and the server runs on the UE machine.

Run the command listed below on the client. Replace 12.1.1.x with the correct IP address of the UE, as assigned by the CN.

```
sudo docker exec ?it oai-ext-dn iperf ?c 12.1.1.x ?u ?b yM --bind 192.168.70.135
```

Run the command listed below on the server. Replace 12.1.1.x with the correct IP address of the UE, as assigned by the CN.

```
iperf ?s ?i 1 ?u ?B 12.1.1.x
```

For the Uplink, the client runs on the UE machine, and the server runs on the CN machine.

Run the command listed below on the client. Replace 12.1.1.x with the correct IP address of the UE, as assigned by the CN.

```
iperf ?c 192.168.70.135 ?u ?b yM --bind 12.1.1.x
```

Run the command listed below on the server. Replace 12.1.1.x with the correct IP address of the UE, as assigned by the CN.

```
sudo docker exec ?it oai-ext-dn iperf ?s ?i 1 -u ?B 192.168.70.135
```

The primary method of technical support is through mailing lists and direct email.

The usrp-users mailing list is for discussions specifically involving the USRP hardware and the UHD software.

https://lists.ettus.com/list/usrp-users.lists.ettus.com

The list archives can be found at the link below.

https://lists.ettus.com/empathy/list/usrp-users.lists.ettus.com

There are two mailing lists, openair5g-user and openair5g-nr, which are for discussions specifically involving the Open Air Interface (OAI) software stack. More information can be found at the links below.

https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/MailingList

https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/AskQuestions

You can contact the authors about the reference architecture directly via email at `support@ettus.com`.