Overview Of SDR Platforms Based On Open Source Software: A 5G System Emulation With Open Air Interface

Emmanouil-Zafeirios G. Bozis
Department of Informatics and Telecommunications
University of Peloponnese
Tripolis, Greece
mbozis@go.uop.gr

Michael C. Batistatos
Department of Informatics and Telecommunications
University of Peloponnese
Tripolis, Greece
mbatist@uop.gr

Nikos C. Sagias
Department of Informatics and Telecommunications
University of Peloponnese
Tripolis, Greece
nsagias@uop.gr

***Abstract*—This paper focuses on the latest trends on software defined radio platforms based on open source software and aims to highlight the benefits that these platforms offer in terms of flexibility and cost. The requirements of hardware and the special features of operating systems that enable the SDR software to run on x86-64 systems are presented in detail. Moreover an emphasis is given to the software architectures used in open source projects implementing wireless protocols. As a paradigm of an open source SDR platform we emulate a 5G system, based on Open Air Interface software and docker engine, installed on a Linux PC. Finally, the main aspects of this emulation are presented and conclusions are made.**

Keywords—Software Defined Radio, 5G, open source software, GNU Radio, Open Air Interface

I. Introduction

For years the implementation of wireless systems in proprietary software and hardware platforms was an obstacle for academia that wanted to experiment on existing and new wireless systems. The introduction of new projects based on open source software and general purpose computers connected to commodity Software Defined Radios (SDR)s, made possible the implementation of fully reconfigurable wireless systems. SDR hardware used in conjunction with open source software like Linux OS and GNU Radio can synthesize a powerful wireless testbed that facilitates the development and testing of new systems. The host PC running the software is connected to SDR hardware via the USB or ethernet port. GNU Radio supports many SDRs, ranging from low cost devices like RTL-SDR [1], HackRF One [2] to more advanced and higher cost devices like bladeRF [3] and Ettus Research USRP [4].

Furthermore the Open Air Interface (OAI) open source software implements the 4G and 5G 3GPP protocols and it can run on commodity computers. The OAI Radio Access Network (OAI RAN) is capable of transmitting and receiving LTE and 5G waveforms with higher cost SDR devices like the USRP B and N series. All baseband signal processing is done in a general purpose x86-64 CPU. Thus it is possible to develop a complete 5G system for private networks or research with commodity hardware and open source software. The development of this system depends heavily on computing technologies like low latency Linux kernels, virtualization, cloud computing and the respective open source software solutions. In the context of the ongoing convergence of wireless communications and computer technologies, it is expected that new open source projects in this field will come out in future. The Linux ecosystem offers flexibility for the development of SDR applications, as there is a plethora of software tools, libraries and code available for the OS. The user can implement a wireless system using all the existing software with no additional cost apart from the cost of hardware. In addition open source software can even be connected to proprietary software and hardware, implementing a part of the system's functionality. Moreover, the GNU Radio environment and SDR hardware provide a practical view for the teaching of theoretical subjects like Communication theory and signal processing to university students. Reference [5] is an alternative methodology in teaching subjects related to signal processing, based on GNU Radio and HackRF One SDR device.

Chapter II presents an overview of SDR platforms based on open source software, chapter III discusses a proposed experimental testbed for 5G system and chapter IV concludes the paper summarizing the results.

II. Open Source SDR Platforms

A. The GNU Radio Platform

GNU Radio is a free and open-source software development toolkit for implementing software radios using signal processing blocks [6]. The developer can use all existing blocks or write new ones in C++ or python language using the software libraries and the underlying framework that connects the signal processing blocks, known as the scheduler. GNU Radio Companion (GRC), the graphical User Interface (GUI) of GNU Radio, helps users in the creation of flow graphs that control the flow of data streams among blocks. The scheduler is the most complicated part of the code base, and it is mainly responsible for the synchronous execution of flowgraph by handling buffered data and their transport from one block to the next, satisfying the block's input/output requirements. The scheduler's role is to help users in the most difficult to program tasks. Thus, users are mainly concerned with writing the code of the block that processes input data and not with the difficult task of transfering data from one block to another synchronously. The existing blocks cover a wide range of signal processing operations, like filtering, FFT transformation, mixing with other signals and interfacing to external SDR hardware. The flow of data starts from the block that has no input called source and ends to the block which has no output called sink. Users can create blocks either as embedded to flowgraphs or as out of tree (OOT) modules that can be imported to other GNU Radio installations. The mathematical operations done

979-8-3503-9958-5/22/$31.00 ©2022 IEEE

are accelerated by Single Instruction Multiple Data (SIMD) parallel processing. The Vector-Optimized Library of Kernels (VOLK) [7] is used to execute SIMD instructions. This library offers a layer of abstraction between the calling function in GNU Radio code and the SIMD code which is specific to the platform/architecture it is being executed in.

While the GNU Radio software can be installed easily on a Linux PC, the installation of the SDR source and sink blocks was a difficult task until recently, mainly due to the existence of different versions of prerequisite software packets in different Linux distributions. PyBOMBS [8], a separate python application can be used in order to overcome the difficulty of installing all software modules like gr-osmosdr in PCs with different versions of software libraries and hardware setup. PyBOMBS detects the user's Operating System and loads all of the prerequisites in the first stage of the build. Then it builds GNU Radio, UHD, and various Out of Tree (OOT) modules from source. The installation is done into a specified user directory rather than in the system paths which caused problems in different linux distributions and software setups. In the current release of Gnu Radio software (3.10 at the time of writing this paper), support for SDRs is made easier with the inclusion of gr-soapy as an in-tree module. This module offers source and sink blocks that support most of the SDR devices, like the AirspyHF, BladeRF, HackRF, LimeSDR, PLUTO, RTLSDR and SDRPlay. Since it is a part of GNU Radio code base, the installation doesn't require any configuration to adapt to the OS environment.

B. Open Air Interface Platform

OpenAirInterface (OAI) is an open source implementation of Radio Access Network (RAN) and Core Network (CN) that gathers a worldwide community of developers who work together to build wireless cellular technologies [9]. OAI is a complete platform for cellular system development as it offers a 4G and Standalone (SA) and Non-Standalone (NSA) 5G stack implementation, compliant to 3GPP standards. OAI includes the CN, RAN and the user equipment (UE). The software supports specific SDR hardware having high speed connectivity to PC through USB 3.0 or ethernet 10 Gbps ports. Among the supported devices are USRP B210, USRP X310, USRP N310 and EURECOM EXPRESSMIMO2 RF card. The hardware requirements of OAI software are high especially in the 5G case. A minimum configuration includes a PC with an Intel core i5 or higher processor with eight or more cores, RAM size of 16 Gbytes and a 10 Gbps network adapter. Currently only Intel processors are supported but in the future other processors are likely to be supported.

The real-time baseband processing in 4G and 5G systems is demanding in computational resources. In order to run the OAI software in a general purpose computer different approaches in system configuration must be taken, like in the CPU operating mode, the Linux kernel, the CPU SIMD instructions and the use of OS level virtualization software. Regarding the CPU operating mode, processor C-States and P-States must be disabled, to keep the CPU frequency stable, by configuring accordingly the BIOS and OS parameters on the host PC. The processor must be always active (C0 state) minimizing the possible latency in the instruction execution. An unwanted consequence of this configuration is the increase in power consumption and thermal dissipation of the CPU. Furthermore, hyperthreading mode must be disabled, since OAI needs to have direct access to the physical CPU cores. In hyperthreading mode the physical cores are divided to logical cores that are treated as if they are actually physical ones by the operating system. Higher layers of the 4G/5G protocol stack like RLC and PDCP use the OAI Inter Task Interface (ITTI) as a full framework to implement an infinite loop on events with inter-thread communication. Lower layers like 5G RLC make direct use of the POSIX thread API. The host PC running the OAI RAN, should have a low latency Linux kernel installed and running. The low latency kernels contain OS level optimizations to achieve the lowest possible latency for time-critical applications like signal processing. The installation of the RAN can not be done in a Virtual Machine (VM), because in a VM the communication through the host ports (ethernet or usb) with the SDR hardware cannot support constant high speed bit rates. This is not the case for the core network (CN), which can run in a VM with a generic Linux kernel. Thus the complete 5G node can be installed in a single host containing a VM for the CN. OAI is compatible only with Intel CPUs, because the optimized DSP functions use SIMD instructions (SSE, SSE2, SSS3, SSE4, and AVX2 versions) that are processor architecture dependent. The software has been tested on Intel i5, i7, Atom and Xeon CPUs. The OAI 5G core network (5GC) is designed with a service-oriented architecture which is adapted to the new service-based architecture (SBA) defined by 3GPP. The network functions (NFs) are separate components that provide services to other NFs and concurrently consume services from them. Additionally, the Control Plane (CP) functions are separated from the User Plane (UP). OS-level virtualization allows the NFs to run on a host as containers which are isolated user space instances. Docker is one of the existing platforms for using containers to build, share and run applications and it is used in the 5G system emulation presented in the next section.

The use of OAI is mainly for experimentation and research and there is no application so far in Mobile Private Networks (MPN). This is related to the fact that general purpose computers are not as power efficient as other solutions like FPGAs and do not offer economic scalability for Mobile Network Operators (MNOs).

III. A Container Based 5G System Emulation On A Single Host Running Open Air Interface Software

We aim to provide information on building a 5G testbed were different scenario testing can be facilitated by running all network services in containers and automating repetitive tasks using the Linux shell. In [10] an emulation of a LTE network was made using OAI modules and OAISIM (Open Air Interface System Emulation). In this paper we extended the work of [10] to a 5G system emulation using docker containers and automated this process by executing a bash script. In a host PC running the OAI software**,** all components of the 5G core network can run inside docker containers as services. In addition to real time operation, OAI 5G RAN includes an RF Simulation mode. The RF simulator is implemented as an OAI device that replaces the actual RF board driver. In this mode instead of using SDR hardware for the transmission and reception of RF signal, the I-Q samples are being transferred directly from the gNB to nr-UE and vice versa via the network. It works like an RF board but it can perform faster or slower than real time depending on CPU speed. When OAI 5G RAN is using the RF simulator mode, the requirements for the real time execution can be relaxed. Thus RAN can be executed inside a deployed container as a service. The same approach can be followed in the deployment of the nr-UE. Consequently with the RF simulator all components of the 5G stack can be executed

inside containers. OAI has uploaded the container images of all NFs to docker hub and provides guidelines on deploying them from their OAI code repository [11]. In this work we retrieved all these images from docker hub. The start up of all images is done with the docker-compose application. The configuration files in yaml format for docker-compose tool are also provided in [11]. We installed in our computer the ubuntu 18.04 LTS OS and all necessary software tools and libraries. Then we managed to run a full OAI 5G full stack system emulation in our PC with one nr-UE connected to gNB as it is depicted in Fig. 1.

In order to automate this process we wrote and executed a single bash script in our host. All necessary modifications to the configuration files were made to match the network setup in our lab. As Fig. 2 shows, the deployment of all containers is done in one host running docker engine. For each container one to three virtual Network Interface Controllers (NICs) are created and IP addresses are assigned.

Furthermore, the public_net and the traffic_net networks are created in the host. These docker networks use the default bridge driver, so they can communicate with each other.

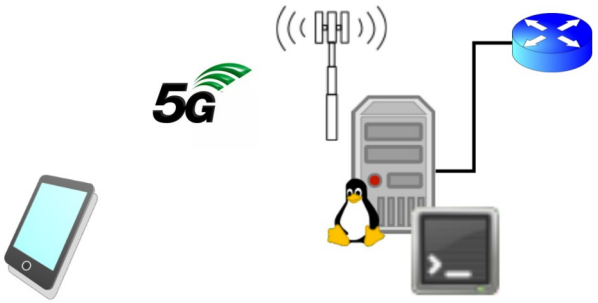

Fig. 1. Simulated 5G scenario

The script starts by deploying the containers of the Network Repository Function (NRF) and mySQL database server. The startup order of containers is controlled with the 'depends_on' property of docker-compose tool. In this way there is no possibility that a NF tries to connect to another NF that has not been deployed yet.

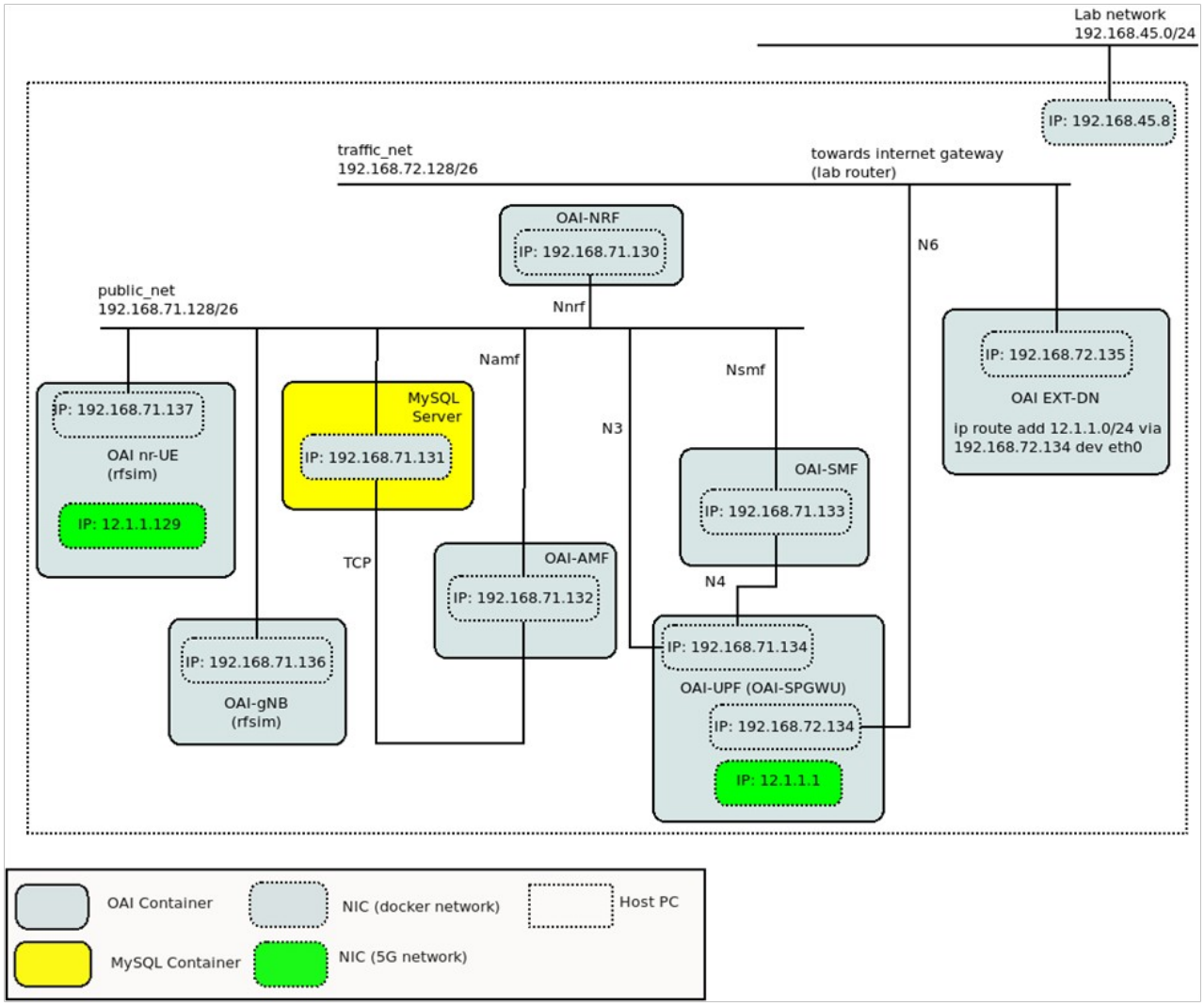

Fig. 2. Diagram of docker containers deployment

The database keeps the subscribers registration data like the International Mobile Subscriber Identity (IMSI) and the Operator Code (OPC). The AMF function retrieves these data and decides upon a received request from a device to attach to the network. The decision is based on comparison of the registration data in the database with the values sent by UE. Then the other containers of the core network functions are deployed. These are the Access and Mobility Management Function (AMF), the Session Management Function (SMF) and the User Plane Function (UPF). The external Data Network (EXT-DN) container is used to route the traffic from the UE to the internet. Between the successive deployment of containers the script pauses execution for a few seconds so that the containers have the necessary time to reach a 'healthy' state. The next container that is deployed is OAI gNB with RF simulator. After the gNB container is up and running, its connection to the AMF function is checked by examining the log of the AMF container. Then the nr-UE container is deployed and the connection of nr-UE to gNB is checked by executing the ifconfig command. The execution of this command reveals if the network interface oaitun-ue1 is in up state. This interface is created after the UE has successfully connected to gNB and all UE data is transferred through this tunnel.

TABLE I. RTT VALUES OF PING COMMAND

Network / IP	***Description***	***RTT (ms)***
etht0 (192.168.71.137)	connected to lab network via docker bridge[a]	0,342
oaitun_ue1 (12.1.1.129)	created by OAI to transport the UE data to 5G network through the GTP tunnel[a]	6,374

[a] See Fig. 2

In order to check the network connectivity through the 5G protocol stack we executed in the nr-UE container ten successive ping commands to the central lab router's IP using the already created oaitun_ue1 interface. We compared the mean value of round-trip-time (RTT) to the value we got executing the ping command using the network interface that is connected to docker bridge (see Fig. 2). The values of RTT are shown in table I. This RTT value is relative to the CPU speed of the host computer running OAI software, because in RF simulator mode the code execution can be done at different speeds. For this reason we didn't take further measurements on throughput, as they would not be accurate for a 5G system that carries out baseband processing in real time.

During the 5G system emulation, we captured the packets through the public_net network with tshark tool. Then we used the wireshark software to analyze the captured packets. We applied a specific filter to display the part of network traffic that is related to 5G protocols and the PING command. Fig. 3 shows a screenshot of the wireshark program displaying captured packets of interest. The first two packets use NG Application Protocol (NGAP) to exchange messages from gNB to AMF for the establishment of the NG Interface. The successful registration of UE to 5G network is seen in the packet exchange from gNB to AMF using the Non-Access-Stratum protocol for 5G System (NAS-5GS). In the captured packets there are also the HeartBeat Request and HeartBeat Response messages exchanged between the SMF and UPF function using the Packet Forwarding Control Protocol (PFCP). The successful execution of ping commands is also noticeable as an exchange of Internet Control Message Protocol (ICMP) packets. When the ping command is executed using the oaitun_ue1 interface the respective ICMP packets are encapsulated in GPRS Tunnelling Protocol (GTP) used as a tranport tunnel for UE data. On the contrary, the ICMP packets created by the ping command using the docker network interface are not encapsulated in GTP as it can be seen in the protocol column in Fig 3.

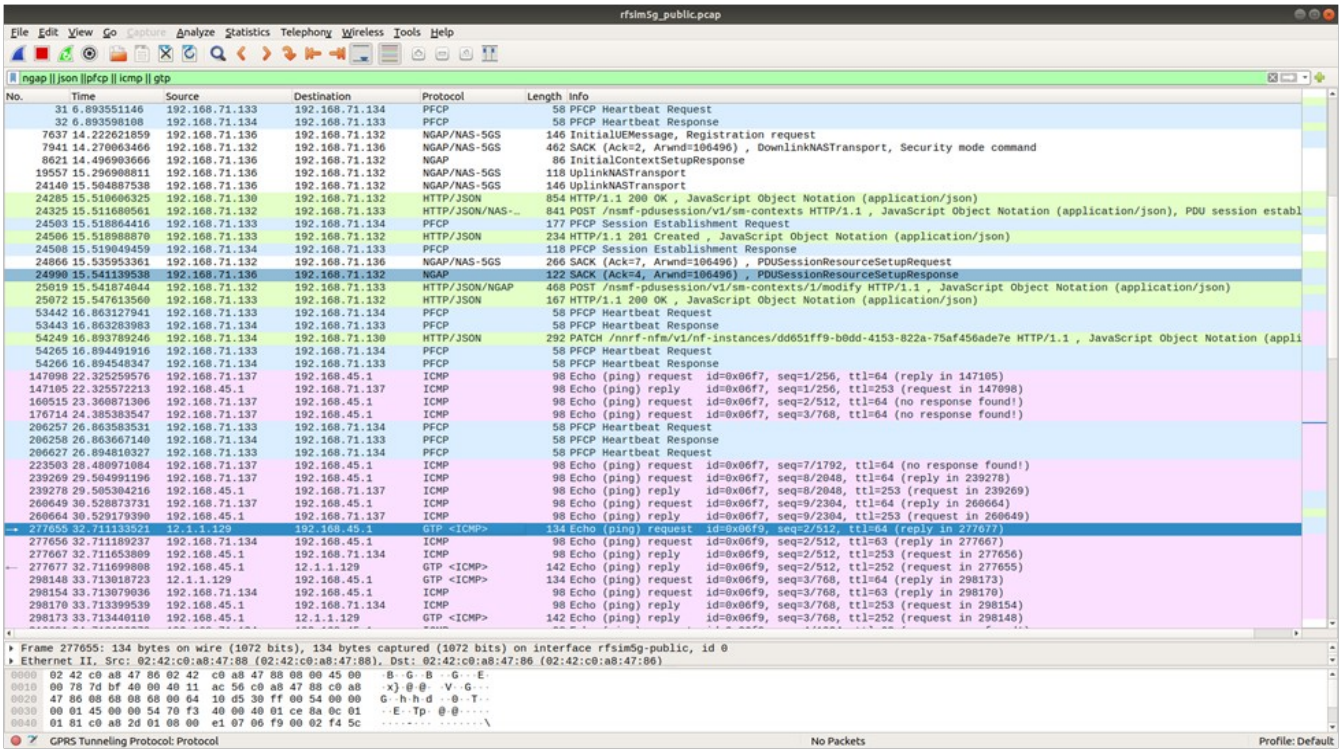

Fig. 3. Captured packets analyzed with Wireshark network protocol analyzer

IV. Conclusions

Open source SDR platforms have significant benefits as they are fully reconfigurable, have lower cost and greater flexibility compared to closed source software solutions. The developer can quickly carry out the programming tasks and configuration necessary to support many existing protocols or to experiment with new ones. In the 5G system emulation with OAI software we had an insight into 5G protocols implementation. We provided information on building a 5G testbed were different scenario testing can be facilitated by running all network services in containers and automating this process using the Linux shell. Furthermore with the help of other open source tools, we managed to analyze the network traffic and control messages exchanged between the 5G NFs.

References

[1] RTL-SDR, https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr, Accessed Sep 1, 2022.

[2] HackRF One, https://greatscottgadgets.com/hackrf/one/, Accessed Sep 1, 2022.

[3] BladeRF, https://www.nuand.com/bladerf-2-0-micro/, Accessed Sep 1, 2022.

[4] USRP Product selector, https://www.ettus.com/products/usrp-product-selector, Accessed Sep 1, 2022.

[5] Del Barrio AA, Manzano JP, Maroto VM, et al. HackRF + GNU Radio: A software-defined radio to teach communication theory. The International Journal of Electrical Engineering & Education. 2019;0(0). doi:10.1177/0020720919868144

[6] About GNU Radio, https://www.gnuradio.org/about/, Accessed Sep 1, 2022.

[7] Volk Library Guide, https://wiki.gnuradio.org/index.php?title=VOLK_Guide, Accessed Sep 1, 2022.

[8] PyBombs, https://github.com/gnuradio/pybombs, Accessed Sep 1, 2022.

[9] Open Air Interface, https://openairinterface.org/about-us/, Accessed Sep 1, 2022.

[10] Nahum, Cleverson & Soares, José & Batista, Pedro & Klautau, Aldebaro. (2017). Emulation of 4G/5G Network Using OpenAirInterface. 10.14209/sbrt.2017.247.

[11] OAI full stack 5G NR RF simulation with containers, https://gitlab.eurecom.fr/oai/openairinterface5g/-/tree/develop/ci-scripts/yaml_files/5g_rfsimulator.