A framework for experimental dense vRAN deployments

Romain Beurdouche
romain.beurdouche@eurecom.fr
EURECOM
Biot, Provence Alpes Côte d'Azur, France

Raymond Knopp
raymond.knopp@eurecom.fr
EURECOM
Biot, Provence Alpes Côte d'Azur, France

Abstract

The advent of 5G virtualized Radio Access Networks (vRANs) brings a new challenge with regards to computer architectures. It requires to design computing technologies that provide appropriate performance while maximizing the efficiency and flexibility of networks. Several solutions were proposed relying on general-purpose processors as well as hardware accelerators. This work presents our effort to enable a dense vRAN deployment on top of these computer architectures using the 5G software stack OpenAirInterface. We describe how we adapted the software stack to leverage the capabilities of hardware and how we managed to operate many vRANs on a shared server. We also share our observations on the behavior of the computer architectures and how they affect the vRANs. We finally discuss the limitations of this work and how they can be addressed to implement better vRAN deployments.

CCS Concepts

• **Networks** → **Middle boxes / network appliances**; **Mobile networks**; **Network experimentation**; • **Hardware** → Platform power issues; *Board- and system-level test*.

Keywords

Mobile networks, Network appliances, Network experimentation, Network virtualization

ACM Reference Format:
Romain Beurdouche and Raymond Knopp. 2026. A framework for experimental dense vRAN deployments. In *20th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization (WiNTECH '26), October 26–30, 2026, Austin, TX, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3831662.3844162

1 Introduction

As the concept of virtual and open Radio Access Network (RAN) was flourishing, silicon industrials such as Intel or AMD released several solutions providing the computing capabilities necessary to efficiently implement 5G RANs at the same time as addressing the requirements in term of flexibility and scalability of virtual RANs (vRANs).

Open-source 5G software stacks also turned to implementing vRANs following the guidelines of the Open RAN (O-RAN) Alliance [24]. While FlexRAN [20] stands as a reference design for implementing the O-RAN specifications, other stacks also achieved end-to-end O-RAN compliant deployments: OpenAirInterface [7] or OCUDU [8]. Besides implementing the O-RAN architecture, open-source 5G software stacks should leverage vRAN computer architectures in order to implement relevant experiments with regards to network efficiency in terms of infrastructure and energy costs. FlexRAN already leverages Intel vRAN accelerators and other open-source 5G software stacks as well have to gain these capabilities. They should as well allow to implement many vRAN instances in parallel on a shared computing infrastructure to enable an efficient usage of large computing nodes.

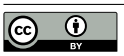
This work aimed to enable the OpenAirInterface stack to leverage three vRAN computer architectures from the industry for implementing efficiently the RAN physical layer. With this stack, we implemented a large deployment of several end-to-end vRANs sharing a single server. We observed the behavior of our deployment and identified necessary improvements to implement further and better scenarios. Our motivation through this effort was to provide a framework for large-scale mobile network deployments that enables full stack experiments in a realistic computing environment [11]. As a first step, Section 2 gives elements of background on the vRAN architecture and its requirements for computing. It also introduces three target systems, each featuring a different computer architecture. Section 3 explains the changes we contributed in the OpenAirInterface stack in order to enable it to leverage the three computer architectures and to enhance the performance of the physical layer. Section 4 introduces a challenging vRAN deployment, explains how we implemented it on the three target systems and presents our observations. The deployment consists in several 5G FR1 small-cells sharing a single host system. Finally, in section 5 we summarize the capabilities and limitations of our work and discuss further improvements.

2 Background

5G introduced new degrees of freedom in the use cases and architectures of RANs in term of network size, range or performance, especially with the advent of private mobile networks. It implies new requirements in RAN solutions in term of flexibility, scalability and ability to be virtualized [12]. Then, silicon industrials released alternative new computer architectures to implement the 5G RAN while virtualizing and sharing the infrastructure. They rely on a variety of building blocks including General Purpose Processors (GPP) as well as Hardware Accelerators (HA). To be more specific, these architectures are designed for implementing the High-PHY layer of the RAN stack in the O-RAN standard architecture.

2.1 vRAN High-PHY

In the O-RAN architecture, the RAN protocol stack is scattered across several network functions. The Radio Unit (O-RU) on the cell site implements the lower physical (Low-PHY) layer while the Distributed Unit (O-DU) located on edge cloud locations close to



WiNTECH '26, Austin, TX, USA

ACM ISBN 979-8-4007-2879-2/26/10
https://doi.org/10.1145/3831662.3844162

the cell sites implements the upper physical layer (High-PHY) and some other layers on top. It implies that the RAN stack is split at a point known as the 7.2 split where a standard network interface known as the O-RAN 7.2 fronthaul [22] ensures the connection of the O-DU and O-RU and of the two halves of the physical layer. High-PHY is the part of the RAN stack located between the top of the physical layer and the 7.2 split. By seating in an edge location, the O-DU is the first network function up from the cell site that can be virtualized in an edge cloud infrastructure. An O-DU can share the edge cloud with other O-DUs, other user-plane network functions and local end services. In an O-DU, the High-PHY layer consists of a digital signal processing pipeline with an uplink (UL) and a downlink (DL) direction. Channel coding is a part of High-PHY that relies on Low-Density Parity-Check (LDPC) codes [31] to fix transmission errors introduced by the channel. Its UL component, channel decoding is the dominant computing consumers in UL High-PHY. Its DL counterpart, channel encoding is one of the dominant computing consumers in DL High-PHY together with precoding, which reckons DL data per antenna port to generate radio beams [18].

2.2 The Architectures

The state of the art. GPPs are able to handle massive Multiple Input Multiple Output (MIMO) broadband processing in the Frequency Range 1 (FR1) spectrum, as demonstrated by Agora [13] and Hydra [19], and 2T2R MIMO in the Frequency Range 2 (FR2) spectrum with tighter time constraints, as demonstrated by Agora-UHD [26]. On the other hand, HAs can also be integrated in a RAN while allowing virtualization provided that HAs interface with software through an appropriate standard interface specified in the O-RAN standard: the Accelerator Abstraction Layer (AAL) [23]. The most common implementation of the AAL is the BaseBand Device (BBDev) application of the Data Plane Development Kit (DPDK) which acts as a performant real-time driver for a variety of HAs and software libraries [17]. For example, Savannah demonstrated this architecture with the FlexRAN stack and an Intel HA for implementing a full-stack 2T2R MIMO FR2 RAN [27]. DecodeX [28] provides a comprehensive comparison of GPP and AAL performance at LDPC channel decoding.

Our infrastructure. Three alternative computing architectures were at our disposal. They were all implemented by silicon industrials and released as commercial products or as engineering samples.

1st High-Performance General-Purpose Processor. The simplest architecture relies on a general-purpose processor with a high count of high-performance cores. It is named hereafter High-Performance General-Purpose Processor or with the acronym **HP-GPP**. In this study we use a processor from the AMD EPYC 9005 series, an AMD EPYC 9575F, with 64 Zen5 cores able to operate at clock frequencies up to 5 GHz and featuring AVX512. This processor has 48 kB of L1 data cache, 32 kB of L1 instruction cache and 1 MB of L2 cache per single core as well as 32 MB of L3 cache per group of 8 cores [3]. The system has 128 GB of DDR5 memory configured with a transfer speed of 6000 MT/s. An Intel E810-C-Q2 QSFP Network Interface Card (NIC) is interfaced via a PCIe gen4 x16 interface. The system runs Ubuntu 25.04 with kernel 6.14.0-1011-realtime as host operating system.

2nd Efficient General-Purpose Processor & RFSoC. The next candidate architecture features a HA for offloading channel coding with AAL. The HA is an RF System-on-Chip (RFSoC) on a PCIe board, a T2 telco accelerator card from AMD [1]. DPDK BBDev is used to interface the HA with software. In choosing the processor of the host, we try to leverage the fact that the processor is relieved from channel coding by the RFSoC. AMD offers two architectures within its 4th generation of EPYC processors, one series focuses on high-performance — the 9004 series — with similar processors as the one we used for the HP-GPP system, and another series focuses on efficiency with regard to silicon and energy consumption — the 8004 series —. We chose a processor from the latter series, an AMD EPYC 8534P, with 64 Zen4c AMD64 cores reaching clock frequencies up to 3.1GHz and featuring AVX512. This processor has 32 kB of L1 data cache, 32 kB of L1 instruction cache and 1 MB of L2 cache per single core as well as 16 MB of L3 cache per group of 8 cores [2]. The system has 128 GB of DDR5 memory configured with a transfer speed of 4800 MT/s. The T2 Telco Accelerator Card is interfaced via PCIe gen3 x16. An Intel E810-C-Q2 QSFP NIC is interfaced via PCIe gen4 x16. The system runs Ubuntu 26.04 with kernel 7.0.0-30-realtime as host operating system. This solution is named hereafter Efficient General-Purpose Processor & RFSoC or with the acronym **E-GPP & RFSoC**.

3rd vRAN System-on-Chip. Another architecture competing with the one above offers to embed the HA in the package of the host processor instead of having it on a separate RFSoC. The HA does not have its own dedicated memory like an RFSoC on a PCIe board but uses the main system memory for data aggregation over repeated transmissions for channel decoding. We have at our disposal an Intel processor from the Sapphire Rapids Edge Enhanced series, an Intel Xeon Gold 6433N, with 32 AMD64 cores reaching clock frequencies up to 2.7GHz and featuring AVX512 and an HA for channel decoding and encoding with the LDPC code. The HA featured in this processor is marketed as 'vRAN Boost'. DPDK BBDev is used to interface the HA with software [15]. This processor has 48 kB of L1 data cache, 32 kB of L1 instruction cache and 2 MB of L2 cache per single core as well as a 32 MB L3 cache shared by the 32 cores. The system has 128 GB of DDR5 memory configured with a transfer speed of 4400 MT/s. An Intel E810-C-Q2 QSFP NIC is interfaced via PCIe gen4 x16. The system runs Red Hat Enterprise Linux 10.1 with kernel 6.12.0-124.35.1.el10_1.x86_64 as host operating system. We name it hereafter vRAN System on Chip or **vRAN-SoC**.

3 LDPC coding with AAL

Leveraging the capabilities of the three architectures becomes possible with enhancements to OpenAirInterface High-PHY that we contributed during the past two years. Our major contributions were merged in the mainstream stack from tag '2026.w35' of August 2026.

As explained in section 2, channel coding with the LDPC code is one of the most important computation consumers in High-PHY UL and DL. Then, one of our main contributions to the stack was

to leverage the capabilities of the vRAN computer architectures to implement efficiently channel coding. We enabled the offloading of channel coding to an HA with AAL. We also improved the efficiency of the offloading by reworking the integration of channel coding libraries in the OpenAirInterface stack.

3.1 Offload to a HA with AAL

In 5G LDPC coding, the Transport Blocks (TB) sent to or received from each User Equipment (UE) within a slot are segmented into code segments also known as Code Blocks (CBs) to be encoded or decoded [31]. Any AMD64 core tend to require a long time to perform the channel decoding even of a single CB. Then GPPs tend to struggle to provide low processing times for channel decoding even by involving many cores in the operation. Offloading channel coding and more especially channel decoding to an HA is therefore rather interesting.

An AAL compliant offload of channel encoding and decoding with LDPC is now available with the OpenAirInterface stack in the form of a shared object that can be optionally compiled and loaded [6]. It relies on DPDK BBDev and 3 HAs are supported:

- **T2 telco accelerator** An RFSoC on a PCIe board from AMD featuring 8 Soft-Decision Forward Error Correction (SD-FEC) cores [1] that we have on the E-GPP & RFSoC system.
- **ACC 100** An accelerator from Intel. We do not own this accelerator but it is operated and tested by another contributor to the stack.
- **vRAN Boost** An accelerator from Intel, also known as ACC 200. It is embedded in the Intel Sapphire Rapids Edge Enhanced processor [15] of the vRAN-SoC system.

BBDev leaves the possibility to the HAs to implement some capabilities or not in order to reflect the actual capabilities of the hardware. Then, the integration of BBDev in the OpenAirInterface stack had to accommodate the respective capabilities of the HAs.

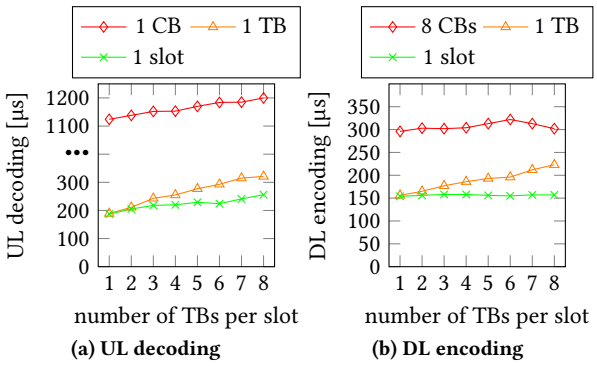
The main difference in capabilities we had to address was that the AMD T2 and Intel ACC 100 have an internal memory while vRAN Boost, which is embedded in a processor, doesn't. A memory is necessary for aggregating data across retransmissions in order to fix transmission errors during channel decoding. With vRAN Boost, aggregated data had to be stored in the host system memory.

Another, more surprising, difference between the HAs was the support of Code Block and Transport Block interfaces. BBDev offers interfaces to process both TBs and CBs, and HAs are supposed to implement both interfaces. In reality, not only the HAs do not implement the two interfaces, but they do not implement the same interface in some cases. While the AMD T2 implements only the CB interface in any case, both Intel ACC 100 and vRAN Boost implement the CB interface in most cases but users must use the TB interface when a TB breaks down in only one CB.

3.2 Implementation interface rework

In order to enable an efficient offloading of channel coding to HAs with BBDev, we reworked the interface to LDPC coding implementations in the OpenAirInterface stack.

Depending on the modulation and coding and on the number of transmission layers, the number of CBs processed in one slot can vary from 1 to 144 for a 100MHz 4T4R RAN. The interface of the



(a) UL decoding **(b) DL encoding**

Figure 1: channel coding time on the E-GPP & RFSoC system with 1 to 8 TBs per slot for different coding function interfaces

channel coding implementations was previously processing 1 CB per call to the decoder and 8 CBs per call to the encoder. One call to decoder or encoder was performing only LDPC coding in the strict sense, which does not include rate matching.

This design was fitting the original GPP based implementation of OpenAirInterface but was too limited to enable an efficient implementation with BBDev. The interface for LDPC coding of BBDev includes LDPC coding and rate matching. The reworked interface of LDPC coding includes code rate matching and enables the processing of all the CBs of one slot in one call to the decoder or encoder. Then BBDev has all the CBs of the slot at its disposal to organize their offloading and processing in an efficient manner.

The gain in term of processing time with this change of the LDPC coding implementations interface is illustrated by Figure 1a and Figure 1b. The figures show the average processing times for the channel decoding and encoding of one slot in a 100MHz numerology 1 RAN operating with 2 UL layers, 4 DL layers and a modulation order of 8. These times were recorded on the E-GPP & RFSoC system. There is one plot for each interface to the LDPC coding implementation that are encountered in the OpenAirInterface codebase history:

(1) The former interface that processes 8 CBs per call for encoding and 1 CB per call for decoding
(2) An intermediate interface enabling to process one TB per call to either encoding or decoding
(3) The newer interface enabling to process the entire slot in one call to either encoding or decoding

One slot breaks down into 63 CBs for 1 TB that occupies the whole slot. The Physical Resource Blocks (PRBs) of the slot are shared between 1 to 8 TBs. Then, these graphs enable to observe the overhead of breaking a slot down into multiple TBs with the three interfaces, knowing that when a slot is shared between multiple UEs, each UE gets its own TB. For both decoding and encoding, there is an improvement with the third interface compared to the first one. The improvement remains whatever the number of TBs in the slot. Meanwhile, with the second interface, the improvement that is seen for one TB per slot decreases with the number of TBs in the slot. Indeed, increasing the number of TBs increases the number of function calls required even though the amount of data to process remains the same. Then, we can conclude that the reworked interface enables significant processing time reductions for channel

encoding and decoding whatever the number of TBs per slot. One can notice that even with the new interface the time to perform LDPC decoding slightly increases with the number of TBs. This is a normal behavior because smaller TBs require more decoding rounds as they profit less from interleaving [31] to compensate the effect of local noise.

4 Dense deployment

We now demonstrate how to challenge the enhanced OpenAirInterface stack on top of each of the three alternative systems with a dense vRANs deployment. This deployment enabled us to determine the efficiency we can achieve with each of the systems in the same networking scenario and observe their respective behaviors. We proceeded by deploying many instances of small-cell 5g O-DUs under traffic load. We first determined the minimum set of resources necessary to operate an O-DU on each system. Then, we replicated this O-DU on each system up to the maximum number of O-DUs achievable while keeping them functional.

4.1 Implemented deployment

Our testing infrastructure didn't offer enough radio appliances to accommodate many O-DUs. Fortunately, the OpenAirInterface stack offers a 'phy-test' mode that emulates the connection of a User Equipment (UE). This 'phy-test' mode can be used together with an O-RAN 7.2 fronthaul executing as it would with an O-RU connected but without a physical connection to an O-RU. It only requires an active network interface. Then, each O-DU instance has an O-RAN 7.2 fronthaul, but only one instance connects to a real O-RU. The O-DU instance with a real O-RU enables to assess the network performance with end-to-end tests. The other O-DU instances execute in 'phy-test' mode to create a fair load on the system. All the O-DUs operate with 4 DL antennas and 4 UL antennas, a bandwidth of 100 MHz on frequency band n77 and numerology 1, corresponding to a Transmission Time Interval (TTI) of 500 µs. DL and UL traffic separation is enforced through Time Domain Division (TDD). For each period of 5 slots, there are 3 slots dedicated to DL traffic, one slot shared between DL and UL traffics and one slot dedicated to UL traffic. On the O-DU instances in 'phy-test' mode, the emulated connection of a UE has 4 transmission layers and modulation order 8 in the DL direction and 2 transmission layers and modulation order 6 in the UL direction. Each O-DU instance is executed within a dedicated container. Containerization enforces processor cores partitioning. The cores assigned to the O-DU containers are isolated in the kernel parameters so that they are not considered for scheduling concurrent processes. On each system, a single NIC is shared by all the O-DUs for their fronthauls by using Single Root Input & Output Virtualization (SR-IOV) and the VFIO driver for DPDK [16]. The SR-IOV mechanism also enables the O-DUs to share the HAs of the E-GPP & RFSoC and vRAN-SoC systems. On the E-GPP & RFSoC system, an administration application should also be executed on a dedicated isolated core in order for several O-DUs to access the RFSoC.

We evaluated the ability of the systems to provide a target mobile-broadband Quality of Service (QoS) with end-to-end tests with a physical UE and the O-DU with a physical O-RU. The end-to-end DL throughput should reach 1200Mb/s with 4 layers and a modulation order of 8 and the end-to-end UL throughput should reach 90Mb/s with 2 layer and a modulation order of 6.

4.2 Resource Assignment

4.2.1 Resource requirements. OpenAirInterface required to assign some processor cores to given tasks. These cores should be isolated to avoid competition with other processes [5]. The roles of these 6 cores are:

- 2 cores for DPDK worker threads for O-RAN 7.2 fronthaul and AAL, namely the DPDK 'IO' core and 'worker' core.
- 1 core for DPDK management, namely the DPDK 'system' core.
- 1 core for managing radio data income and outcome, namely the 'RU thread' core.
- 2 cores dedicated to the High-PHY routines that do not require multi-threading, 1 for UL and 1 for DL, namely the 'L1 RX thread' and 'L1 TX thread' cores.

The O-DU also required a set of cores not overlapping with the 6 cores already assigned. This set of cores known as the 'thread pool' executes worker threads implementing some of the most demanding High-PHY workloads like precoding or resource mapping. It also executes channel coding on the HP-GPP system only. The size of this pool was of 7 cores for a 100MHz 4x4 FR1 O-DU in reference OpenAirInterface setups [4]. Then, each O-DU of our scenario required 13 cores according to reference materials.

For optimizing the processor resources usage in our deployment, we reduced the number of required processors to 8 on the three target systems. First of all, we observed that among the 6 cores statically assigned to single tasks the DPDK 'system' core and 'RU thread' core had a very low level of usage. Therefore, we made two cores of the 'thread pool' overlap with these two assigned cores without introducing any issue. We also tried to reduce the size of the thread pool down to the minimum size enabling the O-DU to remain functional. We managed to reduce the thread pool to 4 cores on the three target systems. Our final processor cores assignment was then:

- 4 single task cores: DPDK 'IO', DPDK 'worker', 'L1 TX thread' and 'L1 RX thread' cores.
- 2 cores shared between the thread pool and tasks assigned to these cores: the DPDK 'system' and 'RU thread' cores.
- 2 cores for thread pool only.

4.2.2 Computer architecture constraint. Beyond the minimum resources required by one O-DU, another constraint was added by the internal architecture of the AMD EPYC 9005 and 8004 processors. These processors are manufactured from 8-cores core complexes with one L3 cache per core complex. 1 (EPYC 9005) or 2 (EPYC 8004) core-complexes are manufactured on a die and dies are assembled in a package to make the processor. Communication between the dies and with other components is ensured by an additional IO die [2, 3]. We observed a significant overhead on the performance when the range of the processor cores allocated to an O-DU crosses the border of a core complex. The behavior is even worse when we cross the border between two dies. In order to avoid this overhead and obtain optimal performances, an O-DU has to be contained on one core complex.

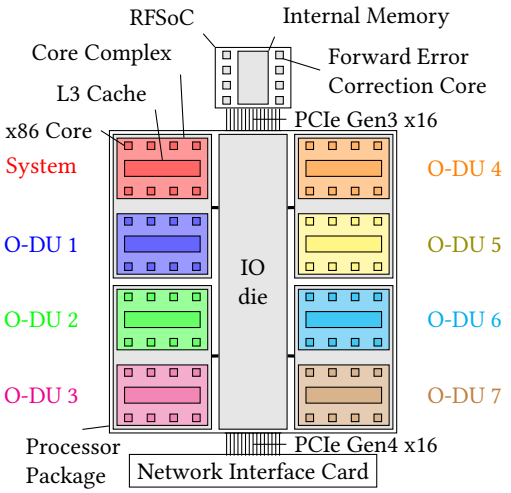


Figure 2: Architecture of the E-GPP & RFSoC system with envisioned O-DU instances placement

This behavior affects the HP-GPP and E-GPP & RFSoC systems. On the vRAN-SoC system, we do not observe such a behavior.

4.2.3 Assignment. The processor of each system is partitioned so that each O-DU is granted the exclusive usage of a range of processor cores.

On the HP-GPP system and E-GPP & RFSoC system, one 8-cores core complex is assigned to each O-DU. except for one core complex that is left to the operating system. We aim to host 7 O-DU instances on these systems with their 64-cores processors. Figure 2 gives an overview of the architecture of the E-GPP & RFSoC system with the intended placement of the O-DUs. The intended instance placement on the core complexes of the HP-GPP system is identical.

On the vRAN-SoC system, 8 cores are assigned to each O-DU as well. Then we aim to host 3 instances on this system with its 32-cores processor with 8 cores left to the operating system.

We also set the processor core frequency to its maximum, disable the idle states of the processor cores and set their frequency governor to performance.

4.3 Observations

On each of the three systems, we increased gradually the number of O-DUs sharing the system from 1 to the number of O-DUs we intended to fit and observed how the network behavior evolved.

4.3.1 Functionality. We couldn't deploy the expected number of O-DUs on the HP-GPP system. Only 5 O-DUs manage to execute out of the 7 expected. When attempting to execute 6 O-DUs, some of them stop executing before any throughput test is started because of delayed reads on the radio fronthaul reception buffer. This means that with 6 O-DUs sharing the system, the fronthaul process becomes too slow to read received radio data on time.

Meanwhile, on the E-GPP & RFSoC system, 7 O-DUs execute out of the 7 expected and the UL and DL throughput tests are successful. On the vRAN-SoC system as well 3 O-DUs are successfully executed out of 3 expected and the UL and DL throughput tests are successful.

4.3.2 Channel coding. Figure 3a and Figure 3b show, for the three systems, the distribution of processing times for LDPC decoding and encoding of a slot recorded over 3 seconds on the O-DU connecting to a physical O-RU under throughput tests with 1 O-DU and the maximum number of O-DUs achieved on the system.

UL channel decoding. For UL channel decoding and one single instance on the system, the HAs outperform the HP-GPP. The fastest solution at decoding is the RFSoC of the E-GPP & RFSoC system. But the picture drastically evolves for a shared system. On the E-GPP and RFSoC system especially, sharing the RFSoC between 7 O-DU instances creates many contention cases that dramatically increase the time to complete the decoding time for most samples. Thus, the huge increase of the median decoding times and other points above in the distribution. It makes that the average decoding time increased from 169µs for 1 instance to 505µs for 7 instances (+199%). The maximum increased from 215µs to 1321µs (+514%). On the vRAN-SoC system, this effect can also be observed to a more limited extend with a moderate increase limited to the higher points of the decoding time distribution. The average decoding time only increased from 271µs for 1 instance to 302µs for 3 instances (+11%) while the maximum increased from 323µs to 661µs (+105%). Meanwhile, the decoding times on the HP-GPP system slightly increase but the distribution keeps almost the same shape. The average decoding time increases from 283µs for 1 instance to 319µs for 5 instances (+13%).

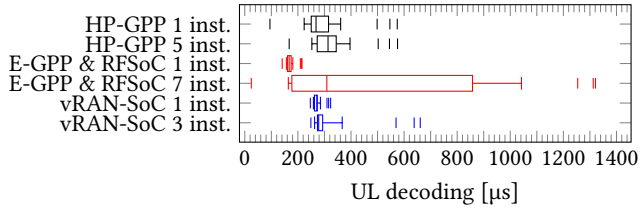
DL channel encoding. For DL channel encoding with a single instance, the HAs are slightly outperformed by the HP-GPP, but the encoding time remains close between the systems. With multiple instances, we observe similar behaviors on the systems with HAs as we observed in the UL case but in a more limited extend. On the E-GPP & RFSoC system, while the increase of the median is rather limited, the higher percentiles significantly increased. The average of encoding times increased from 114µs to 143µs (+25%) and the maximum from 192µs to 797µs (+315%). On the HP-GPP and vRAN-SoC systems, the increase is much more limited and localized in the higher values of the encoding time distribution. For instance, the maximum encoding time only increases from 130µs to 177µs (+36%) on the HP-GPP system and from 119µs to 144µs (+21%) on the vRAN-SoC system.

HA sharing. As already stated, the increased channel coding times on the systems with HAs is the result of contention between O-DUs for using the HA. It is a well-known issue of shared accelerators and a well-studied topic especially when it comes to solutions to mitigate the risk of real-time deadline infringement that this issue creates. Recent studies on this topic are referenced in section 5.

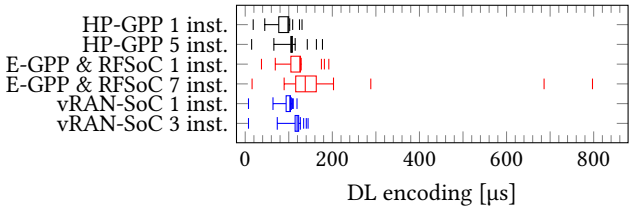
But, in order to understand the true impact of the behavior of these shared systems on the QoS, we need first to look at the impact of these behaviors on the total processing time of the High-PHY layer which comprises other components than channel coding.

4.3.3 Global High-PHY computing. Figure 4a and Figure 4b show the distribution of the total times for processing a slot in the High-PHY layer of the O-DU with real radio with 1 O-DU and the maximum number of O-DUs on the system under the UL and DL throughput tests.

UL High-PHY. In the UL direction, the high-performance processor of the HP-GPP system enables it to show in any case lower processing times than the other systems. The E-GPP & RFSoC remains close to the performance of the HP-GPP system for one instance.

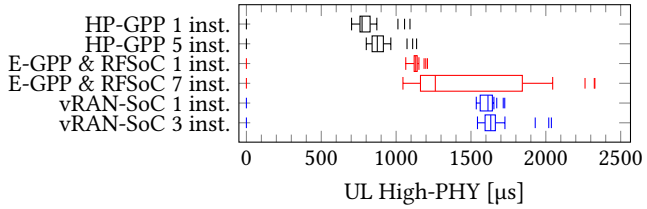


(a) UL decoding

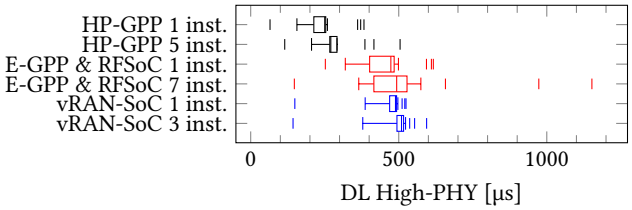


(b) DL encoding

Figure 3: Distribution of time for channel coding of a slot. Whiskers are first and last deciles. Fliers are minimum, 99th percentile, 99.9th percentile and maximum.



(a) UL



(b) DL

Figure 4: Distribution of High-PHY processing time of a slot. Whiskers are first and last deciles. Fliers are minimum, 99th percentile, 99.9th percentile and maximum.

But in the case where 7 instances share the system, the behavior of channel decoding time that we observed translates in a significant overhead on the total UL processing time. The vRAN-SoC system shows processing times well above the two other systems.

It is important to note that, in any case on the three systems, the total UL processing times remain acceptable for a mobile broadband use case. Indeed, the only real limit on processing time for the network to remain functional is to not be so long as to saturate processing. In the way the network is configured, only one slot is dedicated to UL over a 5 slots TDD period and the shared slot cannot be used for UL while UL processing is managed sequentially by a single dedicated thread at the top level in the software stack. Then, processing of one UL slot shall be completed on average within 5 slots or 2500µs to avoid contention with processing of another UL slot.

DL High-PHY. In the DL direction, we observe as well an advance of the HP-GPP system. Meanwhile, the systems with HAs complete the processing of a DL slot in comparable times except, again, the E-GPP & RFSoC system in the case it is shared, in which case the higher percentiles of the processing time distribution show extremely high values. In that last case, the maximum processing time could be as high as 1038µs which represents an increase of 68% compared to the maximum of 617µs for one instance.

Acceptable total DL processing time are bounded by the timing advance with which DL slot processing is started ahead of the over-the-air time which is configured on our network to 7 slots or 3500µs. Long slot processing times should as well not saturate DL processing. 3 full slots and a part of the shared slot are dedicated to DL per 5 slots TDD period while DL processing is managed sequentially by a single dedicated thread at the top level in the software stack. Then, the processing of a DL slot shall complete on average in less than one slot and a quarter or 625µs in order to not saturate the DL processing thread.

Further use cases. The conclusion would be different if our purpose was to implement a ultra-reliable and low-latency network that can require a guaranteed latency as low as 1ms with a reliability up to 99.999% which does not leave room for absorbing outstanding processing times. A deeper study on the reliability of networks and solutions to the issue within this deployment would be relevant.

4.3.4 Computing resource usage. The High-PHY processing times show a superiority of the HP-GPP system with regards to the other systems when it comes to providing guarantees on completion times and therefore on the dependability of the network. On the other hand, the O-DUs on the E-GPP & RFSoC and vRAN-SoC systems behave satisfactorily with regards to the mobile-broadband performance target we set. In addition, the level of usage of the computing resources is better on the E-GPP & RFSoC and vRAN-SoC systems, on which we managed to host the number of O-DUs we initially expected, while, on the HP-GPP system, 16 cores remain unused with only 5 O-DUs actually deployed out of 7 expected. The 56 processor cores that we expected to use for implementing 7 O-DUs on the HP-GPP system cannot be easily redistributed to implement 6 O-DUs instead as this would require from the set of cores allocated to each O-DU to cross the die borders.

4.3.5 Energy efficiency. The energy consumption of the O-DUs on each system is another decisive metric. We are able to measure the intake electrical power at the power supply on the three systems through a standard Redfish [14] management interface. We measure this consumption with the maximum number of O-DUs and throughput tests ongoing.

The total consumption of the HP-GPP system with 5 O-DUs is 558W which is almost 112W per O-DU. Meanwhile, the power consumption is only 288W for the E-GPP & RFSoC system with 7 O-DUs which is almost 41W per O-DU and 263W for the vRAN-SoC which is almost 88W per O-DU with 3 O-DUs.

4.4 Lessons from the deployment

Although this deployment does not exhaust at all the networking use cases that could be implemented with vRANs and the three systems, it still enables us to learn a lot on the abilities and behaviors of the computing systems in intensive computing conditions and to conclude on their fitness for different use cases.

The combination of targeted acceleration and an efficient GPP in the E-GPP & RFSoC system provides a clear superiority in term of computing resource and energy usage efficiency for implementing a mobile-broadband QoS but suffers from an unpredictable behavior when the computing resource is shared, which would be as is a true problem for implementing ultra-reliable and low-latency networks. Meanwhile, the high-performance processor of the HP-GPP offers a predictable behavior and high performance in physical layer processing which may turn in a true advantage for seamlessly implementing ultra-reliable and low-latency networks. But this performance comes at the cost of a sub-optimal efficiency which disqualifies this solution when more efficient solutions can achieve the target QoS. Finally, the vRAN-SoC system may offer a compromise with a better predictability of the behavior of computing than the E-GPP & RFSoC system at the same time as a better efficiency than the HP-GPP system. But this system is still rather weak especially regarding UL computing. Its performance may still be improved though, as we will discuss.

5 Discussions

Based on the lessons from the deployment we implemented, we can summarize the achievements of this work before discussing its limitations and ideas to bridge the gaps.

5.1 Achievements

We addressed some of the main technical challenges in order to implement an efficient deployment of vRANs using OpenAirInterface, which were:

- To modify the stack in order to use it in a dense deployment of vRANs. It means mostly to support and use efficiently the targeted computer architectures.
- To configure the stack to optimize its usage of the computing resources.
- To execute in parallel several vRANs on each system.
- To record performance indicators that enable to compare different alternative computer architectures and identify behaviors of interest.

This work can be used as a framework or referential for conducting further studies on new computer architectures for vRANs or solutions enabling an efficient usage of computing resources. This framework also enables large deployments of vRANs based on an open-source RAN stack with rationalized computing infrastructure costs. It could be used for implementing medium to large scale experimental networks or campus networks.

Instructions and materials to reproduce this framework are made freely accessible in [11].

5.2 Limitations & Enhancements

The vRAN deployment we achieved still has significant limitations. Some of these limitations can be addressed with further enhancements to the OpenAirInterface stack. Other limitations can be addressed by applying solutions from the literature on top of our framework. Then, this work could become an enabler for bringing solutions for efficient computing in vRANs from lab experiments into full-stack end-to-end networks.

5.2.1 Computing architectures support. First of all, additional computing architectures could be supported or better supported in the OpenAirInterface stack to unlock new perspectives.

vRAN-SoC support. First of all, the vRAN-SoC system would benefit from improvements of UL High-PHY routines other than channel decoding in the OpenAirInterface stack. Indeed, we saw in section 4 that although channel decoding times on the vRAN-SoC system are comparable with other systems, total High-PHY UL processing times are, on the other hand, always rather high. Especially the Maximum Likelihood (ML) demodulation could be significantly sped up with a reworked code that we couldn't demonstrate on the vRAN-SoC system within this work.

Then, further capabilities of the vRAN-SoC architecture could be leveraged by the stack. It includes the Fast Fourier Transform (FFT) function of the 'vRAN Boost' HA and the AVX512-FP16 [25] instruction set extension which provides SIMD instructions to operate on 16 bits floating point samples within 512 bits vectors.

Arm processor & HA. Architectures relying on an Arm64 processor accelerated with a HA are an alternative to AMD64 based architectures. We are working on enabling the OpenAirInterface stack to leverage this kind of architectures, such as NVIDIA's Grace Hopper 200 system which features an Arm Neoverse V2 based processor and a GPU as HA [10].

AMD Versal. The T2 HA of the E-GPP & RFSoC system was discontinued by AMD. An alternative are the newer AMD Versal RF Series SoCs. The Versal RFSoCs feature only LDPC decoding accelerators and no LDPC encoding accelerators [9]. Then, LDPC encoding should be implemented on the host system processor. It renews the question of the most appropriate architecture for the processor of the host between the dense and efficient Zen5c and the high-performance Zen5 [3].

5.2.2 Limitation with regards to virtualization. The OpenAirInterface stack and the deployment we achieved still diverge from the fundamental concept of vRAN in some regards while this concept could enable interesting improvements.

Processor cores. The resources assignment constraints from the OpenAirInterface stack and from the AMD EPYC processor architectures prevent to leave RANs running concurrently on shared processor cores. Concurrency with other vRAN instances or even with other best-effort tasks is not possible.

Cores dedicated to single tasks are a common resort in real-time applications in order to avoid any contention that may lead to missing completion deadlines. On the other hand, solutions have been proposed to address the problem of protecting the real-time behavior of a RAN stack on top of shared processor cores. A vRAN aware scheduler interleaves tasks in a way that vRANs do not lack the computing resources they need at the moment they need them. This is for example the approach offered with Concordia [18].

Another constraint added to preserve the real-time behavior of the OpenAirInterface stack is that processor cores have to be set to their maximum frequency and sleep states disabled. This is harmful for the energy efficiency while it could be avoided with an appropriate control system as offered with RENC [21].

O-RAN AAL. OpenAirInterface could also better leverage the O-RAN AAL. The AAL standard leaves the freedom to implement many different operations or 'profiles' beyond the only channel

coding profile that the stack currently uses, for example the FFT profile of the 'vRAN Boost' HA. AAL also offers the ability to distribute operations to several implementations, either HAs or software libraries. This latter ability can be used to reinforce the reliability of vRAN on a shared computing infrastructure. For instance, CloudRIC [30] provides an industry-grade real-time behavior for several vRAN instances sharing a HA and a pool of GPP cores through AAL with custom RAN Intelligent Controller (RIC) and computing aware MAC scheduler. For the case of GPUs, YinYangRAN [29] provides a sharing mechanism to co-locate a RAN workload interfacing through AAL and best-effort workloads on a GPU.

5.2.3 Networking scenario limitations. This work involved FR1 RANs with numerology 1 which are not as challenging in term of real-time requirements as FR2 RANs with numerology 3 and completion deadlines 4 times shorter. The FR2 use case was implemented by Savannah [27] using AAL and the Intel ACC 100 HA but without considering to share the computing infrastructure. The OpenAirInterface stack supports the FR2 spectrum with numerology 3 and the 7.2 split. Then, it may be possible to carry out a study similar to the present work in the FR2 spectrum in the future.

This work doesn't involve neither massive MIMO RANs. Massive MIMO configurations are currently not achievable end-to-end with OpenAirInterface which at best implements a 8T8R MIMO cell with two 4T4R O-RUs. Enabling Massive MIMO with the stack requires to implement digital beamforming for Massive MIMO. Massive MIMO RUs supporting the O-RAN 7.2 fronthaul are also hard to find.

6 Conclusion

In this work, we introduced a framework based on the OpenAirInterface 5G RAN stack for experimenting with vRANs in high-efficiency computing conditions. It enables to implement dense and efficient vRAN deployments on top of computing systems optimized for the RAN physical layer and to observe the behavior of these systems in these operational conditions. We demonstrated the abilities of this framework by implementing a deployment of several 5G small-cell O-DUs on top of three alternative computing systems. This framework can be used for the rational implementation of medium to large scale networking experiments as well as for studies on efficient computing for RANs. It could especially enable to bring solutions providing efficient computing for highly-reliable networking into full-stack implementations with plenty of underlying applications.

Acknowledgments

This research is funded by the European Union's Horizon Europe research and innovation program under grant agreement No. 101092598 (COREnext) and by the French National Research Agency under the France 2030 label NF-Plateforme XG ANR-22-PEFT-00011.

References

[1] 2021. *Xilinx T2 Telco Accelerator Card.* Solution Brief. AMD Xilinx. https://www.amd.com/content/dam/amd/en/documents/products/adaptive-socs-and-fpgas/product-briefs/amd-t2-product-brief.pdf

[2] 2024. *4TH GEN AMD EPYC™ PROCESSOR ARCHITECTURE.* White Paper. AMD. https://www.amd.com/content/dam/amd/en/documents/products/epyc/4th-gen-amd-epyc-processor-architecture-whitepaper.pdf

[3] 2025. *5TH GEN AMD EPYC™ PROCESSOR ARCHITECTURE.* White Paper. AMD. https://docs.amd.com/v/u/en-US/5th-gen-amd-epyc-processor-architecture-white-paper

[4] 2026. Docker compose file for the deployments of a 100MHz 4T4R numerology 1 FHI 7.2 gNB. https://github.com/duranta-project/openairinterface5g/tree/2026.w35/ci-scripts/yaml_files/sa_fhi_7.2_vvdn_gnb/docker-compose.yml?ref_type=tags

[5] 2026. OAI 7.2 Fronthaul Interface 5G SA Tutorial. OpenAirInterface Documentation. https://github.com/duranta-project/openairinterface5g/tree/2026.w35/doc/ORAN_FHI7.2_Tutorial.md

[6] 2026. OAI LDPC offload (O-RAN AAL/DPDK BBDEV). OpenAirInterface Documentation. https://github.com/duranta-project/openairinterface5g/tree/2026.w35/doc/LDPC_OFFLOAD_SETUP.md

[7] 2026. openairinterface5g. OpenAirInterface Git Repository. https://github.com/duranta-project/openairinterface5g/tree/2026.w35

[8] [Accessed: 22-April-2026]. OCUDU Ecosystem Foundation. https://ocudu.org/

[9] [Accessed: 23-April-2026]. AMD Versal™ RF Series. https://www.amd.com/en/products/adaptive-socs-and-fpgas/versal/rf-series.html

[10] Romain Beurdouche and Raymond Knopp. 2026. Implementing vRANs on an Arm and GPU system. In *WiMob '26* (Avignon, France). IEEE.

[11] Romain Beurdouche and Raymond Knopp. 2026. Tutorial: A framework for dense vRAN deployments with OpenAirInterface. https://www.eurecom.fr/publication/8918

[12] Leonardo Bonati, Michele Polese, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2020. Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead. *Computer Networks* 182 (2020), 107516.

[13] Jian Ding, Rahman Doost-Mohammady, Anuj Kalia, and Lin Zhong. 2020. Agora: Real-time massive MIMO baseband processing in software. In *ACM CoNEXT '20*. 232–244. doi:10.1145/3386367.3431296

[14] DMTF. [Accessed: 17-April-2026]. Redfish®. Redfish website. https://www.dmtf.org/standards/redfish

[15] DPDK. [Accessed: 17-April-2026]. 7. Intel® vRAN Boost Poll Mode Driver (PMD). DPDK Documentation. https://doc.dpdk.org/guides/bbdevs/vrb1.html

[16] DPDK. [Accessed: 17-April-2026]. 7. Linux Drivers. DPDK Documentation. https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html

[17] DPDK. [Accessed: 17-April-2026]. 8. Wireless Baseband Device Library. DPDK Documentation. https://doc.dpdk.org/guides/prog_guide/bbdev.html

[18] Xenofon Foukas and Bozidar Radunovic. 2021. Concordia: teaching the 5G vRAN to share compute. In *ACM SIGCOMM 2021*. 580–596. doi:10.1145/3452296.3472894

[19] Junzhi Gong, Anuj Kalia, and Minlan Yu. 2023. Scalable Distributed Massive MIMO Baseband Processing. In *USENIX NSDI '23*. 405–417.

[20] Intel. 2025. *FlexRAN.* https://github.com/intel/FlexRAN/tree/c441ed3b3d692457428ee855e9e13d76b0a36167

[21] Anuj Kalia, Nikita Lazarev, Leyang Xue, Xenofon Foukas, Bozidar Radunovic, and Francis Y. Yan. 2025. Towards energy efficient 5G vRAN servers. In *USENIX NSDI '25*. Article 65, 15 pages.

[22] O-RAN Alliance 2025. *Control, User and Synchronization Plane Specification.* O-RAN Alliance. https://specifications.o-ran.org/download?id=952

[23] O-RAN Alliance 2025. *O-RAN Acceleration Abstraction Layer General Aspects and Principles.* O-RAN Alliance. https://specifications.o-ran.org/download?id=969

[24] O-RAN Alliance 2025. *O-RAN Architecture Description.* O-RAN Alliance. https://specifications.o-ran.org/download?id=932

[25] Niall Power and Sindhu Xirasagar. 2022. *Benefits of Virtualizing the Layer 1 in a RAN Stack.* White Paper. Intel. https://builders.intel.com/solutionslibrary/benefits-of-virtualizing-the-layer-1-in-a-ran-stack

[26] Zhenzhou Qi, Zhihui Gao, Chung-Hsuan Tung, and Tingjun Chen. 2023. Programmable Millimeter-Wave MIMO Radios with Real-Time Baseband Processing. In *ACM WiNTECH '23*. 17–24. doi:10.1145/3615453.3616521

[27] Zhenzhou Qi, Chung-Hsuan Tung, Anuj Kalia, and Tingjun Chen. 2024. Savannah: Efficient mmWave Baseband Processing with Minimal and Heterogeneous Resources. In *ACM MobiCom '24*. 1500–1514. doi:10.1145/3636534.3690707

[28] Zhenzhou Qi, Yuncheng Yao, Yiming Li, Chung-Hsuan Tung, Junyao Zheng, Danyang Zhuo, and Tingjun Chen. 2026. DecodeX: Exploring and Benchmarking of LDPC Decoding Across CPU, GPU, and ASIC Platforms. In *HotMobile '26 (HotMobile '26)*. ACM, New York, NY, USA, 127–132. doi:10.1145/3789514.3792034

[29] Leonardo Lo Schiavo, Jose A. Ayala-Romero, Andres Garcia-Saavedra, Marco Fiore, and Xavier Costa-Perez. 2024. YinYangRAN: Resource Multiplexing in GPU-Accelerated Virtualized RANs. In *IEEE INFOCOM 2024*. 721–730. doi:10.1109/INFOCOM52122.2024.10621380

[30] Leonardo Lo Schiavo, Gines Garcia-Aviles, Andres Garcia-Saavedra, Marco Gramaglia, Marco Fiore, Albert Banchs, and Xavier Costa-Perez. 2024. CloudRIC: Open Radio Access Network (O-RAN) Virtualization with Shared Heterogeneous Computing. In *ACM MobiCom '24*. 558–572. doi:10.1145/3636534.3649381

[31] Mody Sy. 2025. Demystifying 5G Polar and LDPC Codes: A Comprehensive Review and Foundations. *TechRxiv* 2025, 0925 (2025). doi:10.36227/techrxiv.174002467.71268212/v2