sensors

MDPI

Article

Assessing the Cloud-RAN in the Linux Kernel: Sharing Computing and Network Resources

Andres F. Ocampo [1,2,*], Mah-Rukh Fida [3], Ahmed Elmokashfi [4] and Haakon Bryhni [1]

[1] SimulaMet—Simula Metropolitan Center for Digital Engineering, 0167 Oslo, Norway; haakonbryhni@simula.no
[2] Faculty of Technology, Art and Design, OsloMet—Oslo Metropolitan University, 0176 Oslo, Norway
[3] School of Computing and Engineering, University of Gloucestershir, Cheltenham GL50 2RH, UK; mrukh@glos.ac.uk
[4] Amazon Web Services (AWS), Seattle, WA 98109, USA
* Correspondence: andres@simula.no

Abstract: Cloud-based Radio Access Network (Cloud-RAN) leverages virtualization to enable the coexistence of multiple virtual Base Band Units (vBBUs) with collocated workloads on a single edge computer, aiming for economic and operational efficiency. However, this coexistence can cause performance degradation in vBBUs due to resource contention. In this paper, we conduct an empirical analysis of vBBU performance on a Linux RT-Kernel, highlighting the impact of resource sharing with user-space tasks and Kernel threads. Furthermore, we evaluate CPU management strategies such as CPU affinity and CPU isolation as potential solutions to these performance challenges. Our results highlight that the implementation of CPU affinity can significantly reduce throughput variability by up to 40%, decrease vBBU's NACK ratios, and reduce vBBU scheduling latency within the Linux RT-Kernel. Collectively, these findings underscore the potential of CPU management strategies to enhance vBBU performance in Cloud-RAN environments, enabling more efficient and stable network operations. The paper concludes with a discussion on the efficient realization of Cloud-RAN, elucidating the benefits of implementing proposed CPU affinity allocations. The demonstrated enhancements, including reduced scheduling latency and improved end-to-end throughput, affirm the practicality and efficacy of the proposed strategies for optimizing Cloud-RAN deployments.

Keywords: Cloud radio access network (Cloud-RAN); functional splitting; vRAN; 5G cellular networks; RT Linux

Citation: Ocampo, A.F.; Fida, M.-R.; Elmokashfi, A.; Bryhni, H. Assessing the Cloud-RAN in the Linux Kernel: Sharing Computing and Network Resources. *Sensors* **2024**, *24*, 2365. https://doi.org/10.3390/s24072365

Academic Editor: Wei Yi

Received: 6 February 2024
Revised: 2 April 2024
Accepted: 6 April 2024
Published: 8 April 2024

1. Introduction

Considered a key enabler for the upcoming generations of mobile systems, the Cloud-RAN architecture presents a compelling evolution of the RAN design [1]. By leveraging both software-defined wireless networking and virtualization technology, Cloud-RAN enables the deployment of multiple vBBUs on centralized edge servers, providing operational and maintenance benefits for mobile network operators.

To manage latency-critical vBBU functions, the host edge server employs a real-time (RT) Operating System (OS) that offers guarantees to meet the specific latency requirements of these tasks. However, substantial costs related to the development, maintenance, and licensing of an RTOS have led to a shift towards adopting Linux—originally designed as a general-purpose OS-for managing RT systems [2,3]. The Linux Kernel incorporates several RT mechanisms such as RTLinux [4], the low-latency patch [5], and the PREEMPT_RT patch [6], enabling it to support diverse RT systems, including vBBUs [7]. Throughout this paper, we refer to a Linux system equipped with these RT functionalities as the Linux RT-Kernel.

Within the Linux RT-Kernel, the vBBU executes a combination of RT processes for latency-sensitive tasks [8], including physical layer functions, alongside non-RT processes

managing latency-tolerant tasks such as control layer functionalities [9]. Despite the RT-Kernel ability to prioritize RT processes, the performance of latency-critical vBBU functions might be potentially impacted by computation-intensive workloads running in the user-space, as well as from non-preemptible Kernel threads, especially when they are scheduled on the same CPUs. Furthermore, in the advent future generation of mobile systems like 6G and the proliferation of IoT devices, the mobile network architecture must support an ever-increasing number of latency-critical applications. This requires an in-depth understanding of resource sharing implications to maintain stringent performance requirements.

While the existing literature suggests that vBBUs can be effectively run on general-purpose servers [10], the detailed operational implications of such shared-resource scenarios remain not fully understood [11]. Our study seeks to bridge this gap by investigating how compute-intensive workloads and Kernel thread activity influence vBBU performance. Moreover, we assess the effectiveness of CPU management methodologies such as CPU affinity [12] and CPU isolation [13], which are established techniques used to enhance the performance and stability of RT systems in general-purpose computing environments [14–17].

This study offers a focused empirical analysis of deploying vBBUs on general-purpose edge servers managed by the Linux RT-Kernel, with a specific emphasis on the performance implications of resource sharing in Cloud-RAN deployment. Although our analysis is concentrated on this niche, it serves as a foundation for further inquiry into various facets of vBBU performance degradation in different operational contexts. Our contributions are twofold. Firstly, we provide an empirical examination of resource sharing for latency-critical systems like vBBUs on general-purpose edge servers, particularly relevant in emerging generations of mobile systems such as 5G and potential 6G architectures [18,19]. Secondly, we investigate the applicability of CPU affinity and isolation strategies in mitigating performance degradation in Cloud-RAN environments managed by the Linux Kernel. This work advances the understanding of vBBU operations in shared environments, crucial for optimizing Cloud-RAN implementations and ensuring consistent vBBU performance across diverse computational settings.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Section 3 discusses the system model used to deploy vBBU on edge servers. In Section 4, we assess the performance of latency-critical vBBU processes in the Linux RT-Kernel when sharing computing and network resources. This section also discusses strategies for mitigating processing interference from collocated workloads. Section 5 evaluates the performance in the Cloud-RAN architecture, focusing on shared computing and network resources among vBBUs. Lastly, Section 6 concludes the paper.

2. Background and Related Work

This section provides an overview of the Cloud-RAN architecture and highlights the execution of vBBUs on MEC servers managed by the Linux RT-Kernel.

2.1. The Cloud-RAN Architecture

The RAN comprises the User Equipment (UE), the air interface, antennas, the Remote Radio Units (RRUs), the BBU, and a network link connecting the RRU and the BBU, known as Fronthaul. The RAN is connected to the Core Network (CN) via a transport network called Backhaul. As depicted in Figure 1, within Cloud-RAN, the BBU is implemented as a software-defined wireless networking application (vBBU). By leveraging virtualization technologies, multiple vBBUs can be deployed on a centralized MEC server, sharing both processing and network resources [1].

The Cloud-RAN architecture presents a new paradigm for RAN design by enabling the sharing of computing and networking resources. Nevertheless, the centralization of vBBUs brings about stringent latency constraints and capacity demands for both the Fronthaul network and the MEC server that houses the vBBUs. In response to these challenges, the 3GPP has proposed a functional split of the vBBU protocol stack [20].

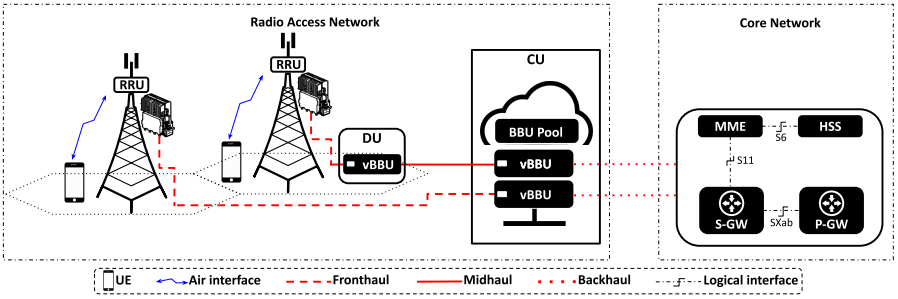


Figure 1. Cloud Radio Access Network architecture.

Processing a portion of the vBBU functions locally near the antennas helps alleviate bandwidth and latency demands on the Fronthaul [21]. This approach is supported by the IEEE 1914 working group that has defined two logical split point placements [22]: the Distributed Unit (DU), near the cell tower, and the Centralized Unit (CU), based on the Mobile Network Operator's edge server. The introduction of these split points reclassified the mobile transport network segments and their respective latency and capacity requirements [23]. The Fronthaul is the segment between the RRU and the DU, the Midhaul connects the DU and the CU with data rate requirements that vary based on the selected functional split, and the Backhaul connects the Cloud-RAN with the CN. Collectively, these transport segments constitute the mobile Crosshaul (Xhaul).

As illustrated in Figure 2, the dotted red line signifies the split option as defined by the 3GPP [20]. In this model, L1 corresponds to physical layer functions (such as low PHY and high PHY), L2 identifies link layer functions (such as MAC and RLC), and L3 is associated with network layer functions (such as PDCP and RRC). Functions to the left of a given option are instantiated at the CU, while functions to the right are allocated to the DU. The more functions designated to the DU, the less stringent the latency and capacity demands.

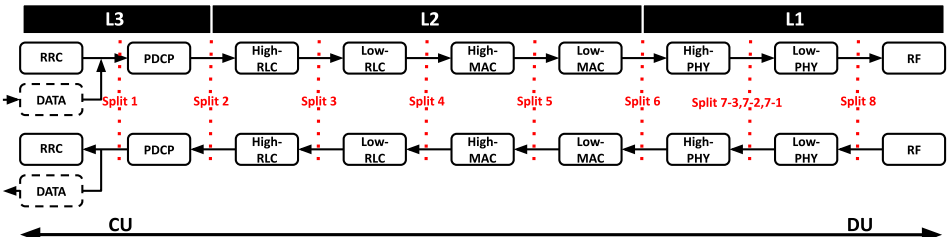


Figure 2. 3GPP functional split of the LTE–BBU functionality [20].

2.2. Hosting vBBU on MEC Servers

While L1 and L2 functions perform signal processing with RT requirements, L3 functions do not have timing demands. Therefore, a vBBU operates as a multi-process application within an OS, composed of a mix of RT and non-RT processes [24]. To meet the timing requirements of the RT processes on a MEC system, the host OS must provide RT guarantees. This includes preemption and a scheduling policy that prioritizes meeting the timing constraints of individual processes over maximizing the average number of scheduled processes.

2.2.1. Linux as the Host OS for Running vBBUs

Over recent years, several mechanisms have emerged to provide RT support within the Linux Kernel. Key examples comprise of RTLinux [4], the low-latency patch [5], and the PREEMPT_RT [6] patch. This development has positioned Linux to be used in RT systems [25], particularly in the domain of RT signal processing for vBBU functions [7]. For instance, Linux RTAI (Real-Time Application Interface) [26] has found applications in mobile system testbeds [7,27–30]. Studies in [29,30] demonstrate the performance of vBBUs using the PREEMPT_RT patch in the context of Cloud-RAN. The Low-Latency

Kernel patch [5], integrated into the mainline code of the Ubuntu distribution, has gained widespread adoption, particularly among researchers using the OAI code [31–33]. This popularity stems from the optimization of OAI's code for seamless compatibility with the Low-Latency Kernel.

2.2.2. Virtualization Environments with RT Support for vBBU Execution

Virtualization technology, such as hypervisors and containers, enables the concurrent execution of multiple vBBUs on the same edge server as isolated processes. In hypervisor-based virtualization, the hypervisor dictates the RT scheduling mechanism that allocates CPU time to virtual machines (VMs). Simultaneously, the guest OS must implement an RT-Kernel capable of preempting non-RT tasks in favor of RT ones [34]. In contrast, for containerized virtualization, the edge server adopts an RTOS with preemption and an RT scheduling mechanism. This scenario requires containers to link their binaries and libraries to the host's RTOS [35].

Containers, as demonstrated in studies like [36], offer superior RT performance compared to VMs, primarily due to reduced overhead from both the hypervisor and the guest OS. For example, investigations in [37,38] explored various virtualization environments within the context of Cloud-RAN, encompassing hypervisors, Docker containers, and Linux containers (LXC). These studies compared them with a bare-metal deployment and found that containers exhibit shorter processing times than hypervisor VMs, with LXC demonstrating processing times similar to those of a bare-metal deployment. Recently in the field of virtualization technology, specifically in containerized virtualization, RT containers [39] have emerged to support RT applications. However, research on the performance outcomes of RT applications running on containers, specifically when sharing resources with collocated workloads, is still required. Likewise, while prior research has validated the feasibility of operating vBBUs on edge servers [32,40], it is critical to understand their performance in scenarios where they concurrently utilize computing resources with collocated workloads [41].

2.3. Resource Sharing in MEC Servers

When running the vBBU on a MEC system managed by the Linux RT-Kernel, vBBU's processes may share computing resources with either collocated user-space processes or Kernel threads. This situation could potentially result in collocated workloads causing processing interference for RT processes. Such interference may stem from the sharing of physical resources [42,43] or Kernel space processing [44,45], both of which could have a detrimental impact on the performance of RT processes [46].

Sharing physical resources, such as CPU, I/O, and memory, among applications with various execution time requirements (e.g., mixed time-critical services—MCS), on general-purpose servers has been extensively studied in the academic literature [47]. However, only a few pieces of research have explored the processing interference resulting from Kernel/user space processing. For instance, a study by Reghenzani et al. in [45] analyzed the processing interference caused by different Kernel subsystems under a variety of workloads related to MCS.

In the context of MCS in embedded systems, a common strategy to mitigate processing interference with collocated applications entails running delay-sensitive services as RT processes on a set of isolated CPUs [43,48]. However, this approach may not align seamlessly with the shared computing resources and multi-tenancy characteristics of Mobile Edge Computing [49]. According to a recent study [50], CPU utilization by a vBBU during peak times does not go beyond 60%, suggesting the possibility of resource sharing. As such, the authors suggested a CPU allocation mechanism for vBBUs that share CPU resources with collocated workloads, intending to minimize CPU underutilization.

Despite these findings, further research is needed to fully understand the performance implications of resource sharing in edge servers, particularly when running RT applications concurrently with other workloads. While existing studies have examined vBBU

performance across different OS and virtualization environments, our research provides a comprehensive investigation by delving into the effects of resource sharing on vBBU performance within the Linux RT-Kernel. This methodology enables us to evaluate not only the overall mobile system performance but also the individual procedures within the vBBU.

3. System Model: Instantiating the vBBU within the Linux Kernel

This section delves into the instantiation of vBBU within the Linux RT-Kernel, discussing both CPU affinity and CPU isolation as strategies to mitigate potential processing interference that may arise from resource sharing with collocated workloads.

3.1. Mobile Network Scenario

Figure 3 depicts the mobile network scenario considered in this paper, which includes a single user equipment (UE), a monolithic vBBU, and the core network (CN). The vBBU is hosted on a MEC server with eight CPUs, managed by the Linux RT-Kernel. The CN, on the other hand, consists of a software implementation of the Evolved Packet Core (vEPC), instantiated on a general purpose server. For detailed software and hardware specifications used in the experimental setup corresponding to the mobile network scenario, please refer to Appendix A.

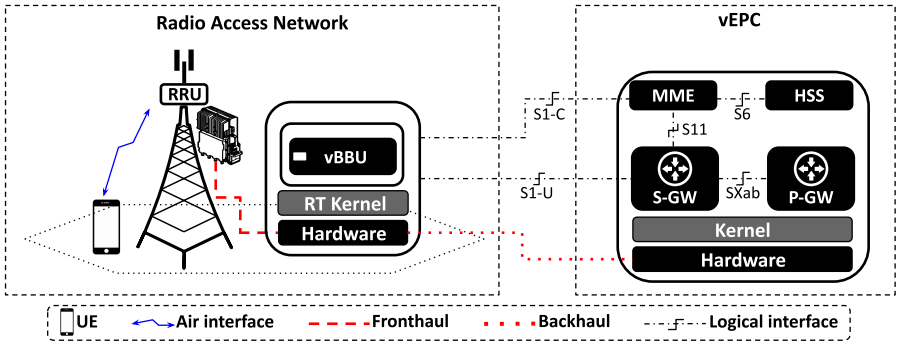


Figure 3. Mobile network scenario: monolithic vBBU running on a bare-metal GPP managed by Linux RT-Kernel. The vEPC is deployed on bare metal GPP managed by Linux generic Kernel.

3.2. Instantiating the vBBU within the Linux RT-Kernel

When instantiated in the Linux RT-Kernel, a vBBU operates as a multi-threaded user-space process, incorporating a mix of both RT and non-RT threads. The method of instantiation for these threads can vary subject to the specific implementation, such as OpenAirInterface (OAI) [51] and SrsLTE [52]. This paper centers around the LTE–eNB implementation by OAI, utilized as the vBBU. This LTE–eNB implementation, when operating under the Linux RT-Kernel, initiates a selection of RT threads and non-RT threads as depicted in Table 1.

The subset of RT threads is responsible for embedding and processing the L1 and L2 functions of the vBBU within the Linux RT-Kernel. These threads are designated as RT user-space threads due to the strict timing requirements associated with their operations. Such strict timing constraints are critical, for instance, in adhering to the Hybrid Automatic Repeat reQuest deadline [32]. In particular, the ru-thread (τ_1) handles low-level L1 functions, primarily dealing with tasks such as reading from and processing signals for the RRU receiver and processing and writing signals for the transmitter. Likewise, the fep_processing thread (τ_3) handles L1 functions through the Front End Process. This involves defining procedures for the Uplink (UL) and processing precoding and Single Carrier Frequency Division Multiple Access signals. For the Downlink (DL), the feptx_thread (τ_4) serves as the FEP for TX and manages DL procedures, pre-coding, and Orthogonal Frequency Division Multiplexing signal processing [40]. Meanwhile, the lte-softmodem thread (τ_2) takes on the processing of the remaining L1 functions along with L2 functions.

Table 1. OAI's LTE–vBBU functions instantiated as threads (subprocesses) in the Linux RT-Kernel.

Thread	Time Requirement	Description	Index
ru-thread	RT	Radio unit processing	τ_1
lte-softmodem	RT	L1–L2 processing	τ_2
fep_processing	RT	Front End Process—RX	τ_3
feptx_thread	RT	Front End Process—TX	τ_4
TASK_GTPV1_U	non-RT	GTP tunneling	τ_5
TASK_UDP	non-RT	UDP socket	τ_6
TASK_S1AP	non-RT	S1 channel	τ_7
TASK_SCTP	non-RT	SCTP channel	τ_8
TASK_RRC	non-RT	RRC channel	τ_9

Conversely, the non-RT thread subset includes L3 processing functions and control procedures such as RRC, STCP, and S1AP. Also categorized as non-RT threads are functions tasked with processing user data plane traffic through the GTP tunnel, an example of which is TASK_GTPV1_U (τ_5). This uses the transport protocol UDP, as seen in tasks like TASK_UDP (τ_6).

3.3. Sharing Computing Resources with User-Space Workloads and Kernel Threads

As the vBBU may share computing resources (e.g., CPU time, I/O, memory) with user-space processes and Kernel threads, understanding the impact of such resource sharing on vBBU performance is crucial. To this end, three vBBU execution scenarios are considered.

The first scenario, denoted *Idle*, involves running the vBBU with no collocated user-space processes or Kernel threads. The second scenario, denoted *User*, entails running the vBBU concurrently with other user-space processes on the same set of CPUs. This setting aims to replicate a situation where the vBBU shares computing resources with non-RT user-space processes. To emulate this, workloads are generated on various subsystems of the edge server. For instance, user-space processes are executed on each CPU, simulating intensive I/O operations, virtual memory stress, disk read/write operations, and message sending/receiving using POSIX. This approach offers a thorough assessment of how the vBBU performs amid the strain of user-space collocated workloads, providing valuable insights into potential challenges and performance implications in real-world scenarios. In the third scenario, named Kernel, the vBBU operates alongside Kernel threads on the same set of CPUs. Kernel threads, serving as entities utilized by the Kernel for CPU-time allocation, are non-preemptive [53]. This characteristic introduces the potential for an impact on vBBU performance if Kernel threads are scheduled on the same CPU [25]. To delve into the influence of Kernel thread processing on vBBU performance, this paper narrows its focus to Hard-IRQs generated when the Linux RT-Kernel receives incoming network packets [54].

Upon receiving a packet, the network interface card (NIC) driver initiates a Hard-IRQ, signaling the availability of an incoming packet for processing. In situations where there is a high flow of incoming packets to the NIC, a corresponding surge in Hard-IRQs occurs. Handling a large number of Hard-IRQs can monopolize CPU time, potentially resulting in starvation for other processes [55]. To address this, the Linux RT-Kernel uses an adaptive mitigation mechanism known as the New API [56]. NAPI suppresses the generation of Hard-IRQs when the number of incoming packets on the NIC exceeds a pre-defined threshold, known as the NAPI weight. As an alternative, NAPI generates software IRQs [57]. These Soft-IRQs systematically poll and process packets in batches, with the batch size matching the NAPI weight. Furthermore, the NAPI's Soft-IRQ handler runs within the context of the NIC's Hard-IRQ handler (i.e., Kernel thread) that triggers the

Soft-IRQ [54]. When the count of received packets drops below the NAPI weight threshold, the NIC's driver disables the NAPI mechanism and resumes the generation of Hard-IRQs.

3.4. Enhancing RT Performance for vBBU Processes with CPU Affinity

The RT-Kernel's Scheduler is designed to maximize the allocation of CPU time across numerous tasks, accomplished by frequently shuffling tasks among available CPUs [44]. However, this dynamic task allocation strategy can introduce processing jitter, causing variability in the wait time for a process before it receives CPU time. Notably, the variability that processing jitter brings about may lead to missed deadlines or unforeseen delays in executing RT functions, potentially compromising the vBBU's overall performance and, consequently, the quality of service in the mobile network.

CPU affinity provides a mechanism to designate which CPUs a particular process can utilize, providing a potential solution to minimize processing jitter [58,59]. By binding RT tasks—such as those within vBBU—to a specific group of CPUs, CPU affinity ensures that the Kernel Scheduler restricts CPU-time allocation to these assigned CPUs [60]. This strategy leverages the principle of processor cache locality, maintaining frequently used data in the cache of the specific CPUs, resulting in a more predictable and deterministic execution of RT tasks [61,62].

For the purposes of our study, we explored three distinct implementations of CPU affinity, each representing a different level of CPU allocation strictness, which confines tasks to specific CPUs, as depicted in Figure 4. Let $C = \{C0, \ldots, C7\}$ be the set of available CPUs in this specific network scenario. Furthermore, let $T = \{\tau_1, \ldots, \tau_9\}$ be the set of vBBU threads shown in Table 1. Assuming CPU affinity allocation to the vBBU on the CPUs $\{C4, C5, C6, C7\}$ of C, the CPU affinity allocation scenarios for the vBBU threads in T are defined as follows:

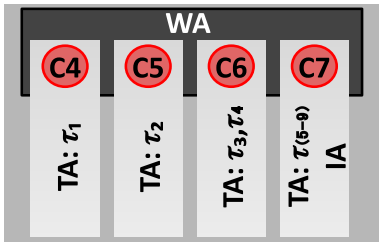


Figure 4. CPU affinity allocation strategies (*WA*, *TA*, and *IA*) for the vBBU.

In the first scenario, named Wide Affinity (*WA*), static CPU affinity is assigned to the vBBU on a predetermined subset of CPUs, denoted as $C_W \subset C$. Specifically, $C_W = \{C4, C5, C6, C7\}$.

In the second scenario, denoted Thread Affinity (TA), static CPU affinity is assigned per process thread (i.e., per child process) on a predetermined subset of CPUs, denoted as $C_T \subset C$. The CPU affinity allocation for each vBBU's thread $\tau_i \in T$ is defined as $C_{\tau_i} \in C_T$ for each $\tau_i \in T$. Specifically, the TA scenario involves grouping related RT threads to run on the same CPU. Given that ru_thread (τ_1) is the most CPU-intensive thread handling time domain IQ signal samples to the RRU, τ_1 is assigned the CPU affinity $C_{\tau_1} = C4$ with no other thread on that CPU. Similarly, as lte-softmodem (τ_2) is the second-most CPU-intensive thread dealing with L1 and L2 functions, τ_2 is given the CPU affinity $C_{\tau_2} = C5$ without any other thread on that CPU. Notably, fep_processing (τ_3) and feptx_thread (τ_4) are related, both performing front-end processing for RX and TX, respectively. Hence, τ_3 and τ_4 share the CPU affinity $C_{\tau_3} = C_{\tau_4} = C6$. Allocating CPU affinity to all non-RT threads on a different CPU than the RT threads helps avoid unpredictable waiting times for the non-RT threads. Therefore, $\{\tau_5, \tau_6, \tau_7, \tau_8, \tau_9\}$ are given by the CPU affinity $C_{\tau_5} = C_{\tau_6} = C_{\tau_7} = C_{\tau_8} = C_{\tau_9} = C7$.

These initial two CPU affinity scenarios aim to explore the benefits of assigning latency-sensitive vBBU threads to dedicated CPUs compared to the alternative approach, where the RT-Kernel dynamically schedules these threads across a predefined set of CPUs.

The third scenario, called Hard-IRQ CPU-affinity (IA), consists of hard-IRQs being assigned specific CPU affinities in order to optimize interrupt handling. Most modern NICs support multi-queue RX/TX. By default, an NIC's driver instantiates as many RX/TX queues as CPUs in the system. By pinning each of these queues to a CPU, the driver registers a Hard-IRQ per queue. An incoming packet is copied to one of these RX-queues according to the traffic flow to which it belongs. Then, the driver raises the corresponding RX-queue's Hard-IRQ or Soft-IRQ to process the packet following the Hard-IRQ or NAPI mechanism.

As demonstrated empirically in [54], assigning a specific CPU affinity to NIC's RX/TX queue Hard-IRQs has been shown to effectively reduce packet processing latency in the Linux RT-Kernel. Building on this insight, we implement CPU affinity allocation to the RX/TX queue Hard-IRQs associated with the NIC port utilized by the vBBU as part of the Backhaul in the network scenario illustrated in Figure 1. Let $Q = \{q_1, \ldots, q_7\}$ represent the set of RX/TX queue Hard-IRQs defined by the NIC's controller, with each of these RX/TX queue Hard-IRQs allocated the CPU affinity $C_{q_i} = C7$, $\forall q_i \in Q$.

The rationale behind allocating CPU affinity to the NIC's RX/TX queue Hard-IRQs on CPU *C7* is twofold. Firstly, this aligns with the CPU affinity allocation to the non-RT thread TASK_GTPV1_U (τ_5) specified in the TA scenario. This strategic alignment not only reduces packet processing latency in the RT-Kernel by pinning the NIC's RX/TX queue Hard-IRQs to a specific CPU but also optimizes the processing of packets consumed by the user-space process [63]. Essentially, τ_5 represents the vBBU's thread responsible for consuming Backhaul packets.

To evaluate the impact of CPU affinity allocation to the NIC on vBBU performance, the IA CPU allocation is integrated with the vBBU CPU affinity allocation in two scenarios: IA-WA, which combines the CPU affinity allocation to the NIC as in IA with the CPU affinity allocation to the vBBU according to WA; and IA–TA, which merges the CPU affinity allocation to the NIC as in IA with the CPU affinity allocation to the vBBU according to TA.

3.5. Mitigating Impact on vBBU RT Performance: Exploring CPU Isolation with Collocated Processes

CPU isolation is a well-known practice in RT systems aimed at preventing process interference [17]. This technique involves dedicating a specific subset of CPUs exclusively to a particular process, ensuring that no other user-space processes, and, in certain scenarios, no Kernel threads, are allocated CPU time on the same subset of CPUs. Implementing CPU isolation in this manner helps RT systems avoid processing interference from collocated workloads, contributing to improved performance and predictability.

We explore two CPU isolation approaches for the vBBU: Shielding (soft isolation) and isolCPU (hard isolation). Shielding involves isolating a specific subset of CPUs from user-space processes not intended to run within the shield. This isolation is achieved through the Kernel's cpuset subsystem [64], which assigns individual CPUs and memory nodes to control groups (Cgroups). Resources allocated to Cgroups are only visible to the Cgroup members or their parents, meaning that certain system Kernel threads may not be moved outside the shield. Thus, Shielding is categorized as a soft CPU isolation approach. Our methodology for studying the vBBU under isolated CPUs using the Shielding approach involves adopting the CPU affinity allocation strategies IA-WA and IA-TA.

In contrast, isolCPU provides a more stringent form of CPU isolation by completely excluding a subset of CPUs from the RT-Kernel Scheduler [65]. Configured at boot time using the isolcpus feature [66], isolated CPUs cannot be allocated to any user-space process or Kernel-space threads, unless the CPU affinity of a process specifies that it should run on the isolated CPUs [17]. Because isolCPU fully prevents the Kernel Scheduler from allocating CPU time to isolated CPUs, it is considered a hard isolation approach [67]. Consequently, only specified processes can run on isolated CPUs, achieved by allocating CPU affinity. The methodology for studying the vBBU when running on isolated CPUs through the isolCPU approach consists of adopting CPU affinity allocations IA-WA and IA-TA.

In this section, we have outlined a comprehensive system model tailored to evaluate resource sharing and CPU management for executing vBBUs on edge servers. Our

analysis incorporated the implementation of CPU affinity and isolation strategies aimed at mitigating the impact of resource contention on vBBU performance. This model serves as the cornerstone for our empirical evaluation, laying the groundwork for the subsequent performance assessments outlined in this paper.

4. Assessing the vBBU Performance in the Linux Kernel

This section conducts an empirical study on the performance of vBBU processes within the Linux RT-Kernel, instantiated as detailed in the system model in Section 3. The primary objective is to investigate the implications of resource sharing. Additionally, we delve into the application of CPU management strategies—specifically, CPU affinity and CPU isolation—with the aim of improving RT performance and mitigating processing interference.

4.1. Methodology

The methodology employed to assess the vBBU performance in the Linux RT-Kernel involves evaluating the vBBU performance for each of the resource sharing and CPU management strategy scenarios described in Section 3.

4.1.1. Experimental Design

A series of experiments were conducted on the mobile network testbed scenario (Figure 3). Table A1 (please refer to Appendix A) summarizes the software and hardware specifications used to deploy this experimental setup. Utilizing the synthetic benchmark tool Iperf3 [68], a TCP flow with a target data rate of 10 Mbps was transmitted from an Iperf3 server situated outside the mobile network to an Iperf3 client at the UE.

For each of the resource-sharing and CPU-management scenarios outlined earlier, we carried out a series of six experiments, each of which lasted 10 min, resulting in a total observation period of 60 min. The duration of each experiment was selected to collect a significant number of data points that would be statistically significant for our analysis.

As outlined in the system model in Section 3, the User scenario is designed to simulate a situation where the vBBU shares computing resources with non-RT user-space processes. To achieve this, we used the synthetic benchmark tool stress ng [69] to generate workloads on various subsystems of the host machine. For example, we executed stress-ng instances on each CPU to perform I/O operations, stress the virtual memory, and write/read on disk (stress-ng -d 1 –hdd-bytes 200M -m 1 –vm-bytes 1G –iomix 1 –iomix-bytes 100M –mq 0.)

In the Kernel scenario, vBBU processes are executed alongside non-preemptive Kernel threads. As mentioned earlier, this scenario considers the processing of Hard-IRQs generated from incoming network packets as a use case for Kernel thread processing. To induce a high number of Hard-IRQs in the RT-Kernel, we introduce high packet-rate background traffic (BT) to the vBBU host edge server. The BT is generated by the Anritsu MT1000A Network Master Pro Tester [70] (referred to as the Anritsu device throughout the paper). Configured as a UDP flow with a frame size of 100 bytes and transmitted at 1000 Mbps (i.e., the maximum rate supported by the host's NIC), BT uses a different NIC port than the one used by the vBBU in the Backhaul network. The host machine receives BT packets through an open UDP port. The Netcat network utility [71] serves as the receiver user-space application for BT packets. In this specific case, Netcat reads and writes packet content into the command line, generating additional workload on the edge server.

Additionally, to assess the effect of allocating CPU affinity to the NIC as defined for the IA scenario (see Section 3 for a system description on the CPU affinity), we conducted experiments to measure the RTT between the edge server hosting the vBBU and the Anritsu device. Specifically, we measured the RTT after sending ICMP packets (at a rate of one packet per second) to the NIC port used in the Backhaul, for which CPU affinity has been defined. The Anritsu device, which provides precise time synchronization for frame-delay measurements, was used to measure the RTT.

The methodology involves evaluating four scenarios. First, the noA scenario represents the default configuration where no CPU affinity has been defined for the NIC port used in the Backhaul. The second scenario, named IA, adopts CPU affinity allocation based on IA for the RX/TX queue Hard-IRQ of the NIC port used in the Backhaul. The remaining scenarios incorporate the BT flow. The inclusion of BT allows us to study latency when the Linux RT-Kernel utilizes the NAPI mechanism. The third scenario, noA-BT, combines the noA configuration on the NIC port used as Backhaul with the BT in a second NIC port. Finally, the fourth scenario, IA-BT, involves allocating CPU affinity based on IA to the Backhaul port, alongside the BT in the second port.

4.1.2. Derived Metrics for Evaluation

To assess the performance of vBBU processes, we employ specific metrics that offer insights into their functionality. Two key metrics are considered for evaluation:

1. DL NACK: This metric is part of the DL Data Transmission Process (HARQ ACK/NACK) [72]. If the UE detects an error in received DL data, it sends a DL NACK to the vBBU, triggering DL data retransmission. NACK counts, which indicate DL re-transmissions, provide valuable insights into the DL data path's health;
2. Scheduling Latency: Measuring the waiting time a process undergoes to obtain CPU-time in the RT-Kernel, scheduling latency is a crucial metric that contributes to task processing latency [41]. Using the Kernel tracing tool BPF Compiler Collection [73], we assessed the scheduling latency for each vBBU thread outlined in Table 1.

4.2. Results and Discussion

Here, we present and analyze the results of our empirical study on vBBU performance within the Linux RT-Kernel, with a specific focus on resource-sharing and CPU-management strategies.

4.2.1. Resource Sharing Impact

First, we analyze vBBU performance across the resource-sharing scenarios described in Section 3—Idle, User, and Kernel. The Idle scenario serves as our performance baseline, providing a reference point where the vBBU operates in isolation, without the influence of collocated user-space workloads or Kernel threads. By comparing the outcomes of the other two scenarios to the Idle scenario, we gain insights into how resource-sharing and CPU-management strategies impact the vBBU's functionality and efficiency.

The DL NACK metric is a crucial indicator of the health of the DL data path, where fewer NACKs imply a more robust path. Figure 5 presents the cumulative count of vBBU's DL NACK for each scenario. Notably, the User scenario, where the vBBU operates alongside collocated user-space workloads, exhibits a 3% increase in DL NACK compared to the Idle scenario. This increase presumably stems from the additional strain on computing resources resulting from the concurrent execution of collocated user-space workloads alongside the vBBU.

In the Kernel scenario, the number of DL NACKs experiences a reduction of 8.6% compared to the Idle scenario. This decrease can be attributed to the efficient handling of high packet rates by the Linux RT-kernel. The Kernel scenario involves processing a high rate of incoming packets as a use case of Kernel thread processing. In the RT-Kernel, the handling of a high rate of incoming packets is achieved using NAPI, which polls and processes packets in batches within the context of Hard-IRQs.

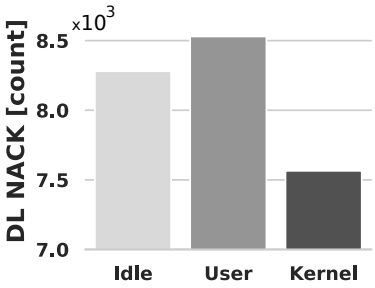


Figure 5. vBBU procedures: number of DL NACKs computed over a set of six experiments (total observation time-span is 60 min). Evaluated scenarios: (i) Idle (no resource sharing); (ii) User (sharing resources with collocated user-space workloads); (iii) Kernel (sharing resources with collocated Kernel threads).

The observation that the Idle scenario yields a higher count of DL NACKs compared to the Kernel scenario suggests that processing packets with a low incoming rate has a greater impact on the vBBU than processing packets with a high incoming rate in the RT-Kernel. In the Idle scenario, the edge server hosting the vBBU processes mobile traffic from the experiment, coming at a rate of 10 Mbps. Given that this traffic has a low enough packet rate to keep NAPI disabled, the NIC controller generates a Hard-IRQ for each arriving packet.

Table 2 provides information on the RT-Kernel scheduling latency events for the vBBU threads described in Table 1. The table categorizes events according to the percentage that falls into different latency buckets. In particular, it includes time intervals up to 15 microseconds, which covers more than 99% of all latency events.

Table 2. Scheduling latency of vBBU's threads (percentage). Evaluated scenarios: (i) Idle (no resource sharing); (ii) User (sharing resources with collocated user-space workloads); (iii) Kernel (sharing resources with collocated Kernel threads).

Thread	0–1 (microseconds)			2–3 (microseconds)			4–7 (microseconds)			8–15 (microseconds)		
	Idle	User	Kernel	Idle	User	Kernel	Idle	User	Kernel	Idle	User	Kernel
ru-thread	93.8	53.1	90.9	0.4	41.9	4.8	0.1	1	0.3	5.7	4	3.9
lte-softmodem	99.4	49.5	91.6	0.3	49.1	7.8	0.01	0.7	0.2	0.3	0.6	0.4
fep_processing	99.3	63	89.7	0.4	35	9.3	0.05	1.1	0.5	0.3	0.9	0.4
feptx_thread	99.4	51.4	76.1	0.5	44.8	23	0.03	3.5	0.7	0.03	0.1	0.1
TASK_GTP	98.6	58.6	84.8	0.8	38.1	10.7	0.4	2	3	0.2	1.1	1.2
UDP_TASK	95.2	52	77.9	1	41.4	12.9	2.8	4.4	3.7	0.8	2	1.8

In the Idle scenario, more than 93% latency events for all vBBU threads fall within the 0–1 microsecond interval. However, in the User scenario, where vBBU shares computing resources with collocated user-space workloads, there is a notable increase in scheduling latency for RT threads. Specifically, there is a shift in at least 35% of scheduling latency events to the 2–3 microsecond interval compared to the Idle scenario.

In the User scenario, where various subsystems like memory, disk, and I/O are stressed, there is a more pronounced impact on scheduling latency compared to the Kernel scenario. However, this impact differs among CPU-intensive threads, such as RT ru-thread and lte-softmodem, versus less intensive RT threads like fep_processing or feptx_thread. This difference can be attributed to their unique activity levels. Specifically, ru-thread and lte-softmodem consume more CPU time allocated to the vBBU compared to the fep_processing or feptx_thread, which have periods of inactivity ranging from 0.5–1 ms. Notably, the inactivity of the ru-thread never exceeds 64 μs. Thus, an increase in latency predominantly occurs during the wake-up of these inactive threads.

In the Kernel scenario, scheduling latency events for all vBBU threads exhibit higher latency compared to the Idle scenario. Even though this contrasts with the effect observed

for NACKs, where the vBBU benefits from the efficient processing of high packet rates using NAPI compared to the Idle scenario, Kernel thread processing introduces a noticeable increase in the scheduling latency of the vBBU. This occurs because Hard-IRQ processing of incoming packets via Kernel threads cannot be preempted by the vBBU, causing prolonged wait times for it to access CPU-time.

Certainly, non-RT threads benefit from having more available CPUs in both idle and stressed systems. Since non-RT threads can be preempted by RT threads in the RT-Kernel, having more CPUs increases the opportunities for a non-RT thread to execute.

4.2.2. CPU Management Strategies

This section assesses two distinct approaches aimed at mitigating the impact of sharing computing resources on the vBBU. Initially, we explore CPU affinity as a strategy to enhance the RT performance of vBBU latency-critical functions. Subsequently, we investigate the advantages of CPU isolation as a method to alleviate processing interference.

CPU Affinity

For each resource sharing scenario (i.e., Idle, User, Kernel), Table 3 displays scheduling latency events of vBBU threads under CPU affinity scenarios WA and TA. The table presents the percentage of events that fall into different latency buckets.

Table 3. Scheduling latency of vBBU's threads (percentage). Evaluated scenarios: (i) Idle (no resource sharing); (ii) User (sharing computing resources with collocated user-space workloads); (iii) Kernel (sharing computing resources with collocated Kernel thread processing-packet processing use case)-CPU affinity allocation: WA vs TA.

Thread	0–1 (microseconds)						2–3 (microseconds)						4–7 (microseconds)						8–15 (microseconds)					
	Idle		User		Kernel		Idle		User		Kernel		Idle		User		Kernel		Idle		User		Kernel	
	WA	TA	WA	TA	WA	TA	WA	TA	WA	TA	WA	TA	WA	TA	WA	TA	WA	TA	WA	TA	WA	TA	WA	TA
ru-thread	93.5	93.3	45.7	52.9	78.8	81.6	0.4	0.3	49.2	41.1	17.2	15.4	0.1	0.1	1.4	1.7	0.7	0.6	5.9	6.2	3.6	4	3.3	2.3
lte-softmodem	99.3	99.1	28.6	49.2	74.2	80.9	0.5	0.3	57.1	49	19.3	17.4	0.03	0	14.3	0.6	0	0.9	0.2	0.1	0	0.8	6.4	0.4
fep_processing	98.8	99.2	45.4	56.7	70.3	60.1	0.5	0.2	46.8	39.7	25	33.8	0.02	0	5.5	1.8	3.1	4.3	0.5	0.6	2	1.7	1.42	1.6
feptx_thread	99.3	99.7	47	53.4	72	54.4	0.7	0.3	49.7	45.6	26.2	42.8	0.05	0	3	0.9	1.7	2.2	0.02	0	0.2	0.1	0.1	0.4
TASK_GTP	94.4	6.4	46.9	2.3	75.6	1.04	0.8	86.3	43.8	89.6	17.9	90.5	3.3	6	4.9	3.3	3.9	3.9	1.3	1	3.8	4.4	2.3	4.3
UDP_TASK	95.4	90.8	22.9	38.7	57.4	51.3	1.4	6.2	65.2	53.5	35.8	40.7	2	1.3	7.8	5.4	4.4	3.4	1	1	3.7	1.5	2.1	3.8

As mentioned earlier, the Idle scenario serves as a performance reference, as no other user-space processes are running on the edge server except the vBBU. Although there is no significant difference in the scheduling latency of RT threads when running the vBBU under WA or TA, non-RT threads experience higher scheduling latency in TA than in WA. The reason is that, in WA, the scheduler grants CPU time to the vBBU on any of the CPUs defined by the CPU affinity C_W. Conversely, when pinned to a fixed CPU, as in TA, a non-RT thread would experience an increased waiting time in the run queue until other tasks on the same CPU are descheduled.

In the User scenario, despite nearly half of vBBU's RT threads experiencing a shift in scheduling latency events from the interval of 0–1 microseconds to 2–3 microseconds, the impact on the vBBU's RT threads is less pronounced in TA than in WA. By pinning RT threads to a specific CPU in TA, these scheduling latency shifts are reduced by up to 8% compared to WA.

In the Kernel scenario, where the vBBU processes incoming packets in the context of Hard-IRQ, the impact on vBBU varies based on its thread characteristics, with critical threads such as ru_thread and lte-softmodem benefiting from TA, resulting in a 2% reduction in shifted latency events from the 0-1 microsecond interval to the 2–3 microsecond interval compared to WA. Conversely, RT threads fep_processing and feptx_thread experience an increase in scheduling latency under TA, with a 9% and 16% higher number of shifted latency events in the 2–3 microsecond interval than in WA, respectively. This increase is attributed to inactive periods experienced by these threads and the act of

waking them up on the specific CPU they are pinned to in TA while Kernel threads are being scheduled.

In summary, when sharing computing resources with user-space workloads, vBBU's RT threads benefit from TA CPU affinity as the impact on scheduling latency is lower than in WA. However, TA benefits only the two most demanding vBBU's RT threads (ru_thread and lte-softmodem) in mitigating the impact of Kernel threads processing from incoming packet in the Linux Kernel. Unlike TA, where threads are pinned to a fixed CPU, WA circumscribes the vBBU's threads to run on any of the CPUs in C_W, according to the Kernel's scheduling policy. As a result, non-RT threads benefit from more available CPUs either on idle or loaded systems. Because non-RT threads might be preempted by RT threads in the RT-Kernel, the more CPUs, the more execution chances possessed by a non-RT thread.

Evidence suggests that allocating CPU affinity to the NIC's RX/TX queue Hard-IRQs reduces packet processing latency in the Linux RT-Kernel [24]. Before analyzing the benefits of CPU affinity allocation based on IA to the vBBU performance, this section first studies the extent to which CPU affinity to the NIC mitigates packet processing latency by measuring the RTT between the edge server hosting the vBBU and the Anritsu device.

For each of the scenarios defined above, Figure 6 shows the RTT distribution. From these results, we make three observations about (i) the benefits of defining CPU affinity for the NIC; (ii) the increase in delay caused by the BT; (iii) the impact caused by per-packet processing through Hard-IRQ. For (i), the analysis is focused on the interquartile range. Defined as the difference ($IQR = Q3 - Q1$) between the upper quartile ($Q3$: 75th percentile) and the lower quartile ($Q1$: 25th percentile) [74], the IQR provides insights into the central 50% of the data. The higher the IQR value, the more spread among the central 50% of the data. In this case, a wider distribution indicates a higher latency variation (jitter) of packet processing in the RT-Kernel. It is worth noting that low jitter and determinism are crucial for RT processes. For instance, the IQR for the two scenarios where no CPU affinity allocation is defined, noA and noA-BT, exhibits a wider spread, reaching up to 0.6.

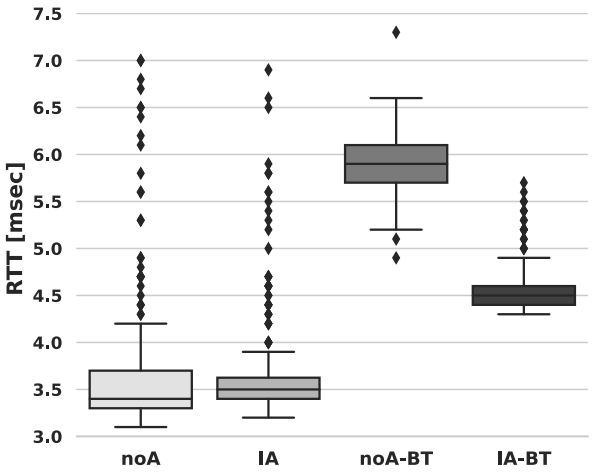


Figure 6. RTT estimate of packet processing latency in the RT-Kernel: comparing CPU affinity allocation to the NIC's port RX/TX queue Hard-IRQ with no CPU affinity.

In contrast, assigning CPU affinity to the Backhaul NIC port yields a more condensed distribution, consequently reducing jitter. The IQRs are 0.33 and 0.3 in IA and IA-BT, respectively. In essence, CPU affinity for the NIC reduces the spread of the data by half for 50% of the samples, compared to scenarios without CPU affinity.

In the presence of BT processing, there is an observable increase in packet processing delay (ii). Specifically, in the case of noA-BT, the RTT distribution experiences an increase of approximately 2.4 msec compared to noA, representing a substantial increase of 68%. Conversely, in IA-BT, the RTT distribution increases by around 1 msec, indicating a 28% rise

compared to IA. This analysis effectively highlights the impact of BT processing on packet-processing delay and demonstrates the benefits of CPU affinity in mitigating this increase.

The boxplot outliers in the RTT distribution provide clear evidence of the impact caused by per-packet Hard-IRQ processing in the RT-Kernel (iii). Outliers, represented by data points outside the whiskers, highlight extreme values in the dataset. Whiskers, defined as the most extreme data points within the range of $Q1$ and $Q3$ and not greater than $1.5 \times IQR$ [8], help identify the presence of outliers.

In scenarios such as noA and IA, where the bit rate is low, preventing the activation of the NAPI mechanism, each packet is processed using Hard-IRQs, leading to unpredictable latency. The outliers observed in these scenarios are indicative of the unpredictable delays, potentially caused by factors like livelocks [75]. Notably, in noA, the outliers represent 9% of the data, while, in noA-BT, where NAPI is enabled due to the high packet rate flow of BT, outliers account for only 0.3% of the data. Similarly, in IA, outliers represent 11% of the data, while, in IA-BT, with NAPI enabled, outliers represent 6% of the data. This observation underscores the effectiveness of NAPI in mitigating unpredictable latency induced by per-packet Hard-IRQ processing.

Certainly, these results suggest that allocating CPU affinity to the NIC enhances the determinism of packet processing in the RT-Kernel. Now, the question is whether the vBBU would benefit from allocating CPU affinity to the NIC port used in the Backhaul. To answer this question, we next evaluate the scheduling latency of vBBU threads while incorporating IA into the CPU affinity allocation scenarios defined for the vBBU: IA-WA and IA-TA.

Table 4 presents the scheduling latency events of vBBU threads under CPU affinity allocations IA-WA and IA-TA. As this analysis centers on adopting CPU affinity allocation to the NIC port used in the Backhaul, we specifically evaluate the Kernel scenario, involving the sharing of computing resources with Kernel thread processing generated by incoming packets.

Table 4. Scheduling latency of vBBU's threads (percentage). Evaluated scenarios: (i) Kernel (sharing computing resources with collocated Kernel thread processing generated by incoming packets)—CPU affinity allocation: IA-WA vs. IA-TA.

Thread	0–1 (microseconds)		2–3 (microseconds)		4–7 (microseconds)		8–15 (microseconds)	
	Kernel		Kernel		Kernel		Kernel	
	IA-WA	IA-TA	IA-WA	IA-TA	IA-WA	IA-TA	IA-WA	IA-TA
ru-thread	89.4	90.8	6.5	4.5	0.4	0.34	3.5	4.3
lte-softmodem	81	93.4	7.2	5.4	0.8	0.2	1.5	0.3
fep_processing	86.3	80.2	11.2	17.7	0.6	0.8	0.8	1.2
feptx_thread	72.9	75.9	15.6	23.4	1.2	0.4	1.4	0.1
TASK_GTP	74.6	0.1	9.3	56.4	4.7	32.6	2.2	4.5
UDP_TASK	62.5	49.4	6.2	2.7	23	32.5	3.5	14.2

These results indicate that the vBBU benefits from allocating CPU affinity to the Backhaul port in mitigating the impact of packet processing in the Linux RT-Kernel. In both scenarios, IA-WA and IA-TA, scheduling latency events of vBBU RT threads shows an increase in the 0-1 interval compared to the results reported in Table 3, where no CPU affinity was allocated to the NIC. In other words, adopting IA to the Backhaul port helps alleviate the impact on the vBBU RT threads scheduling latency caused by packet processing.

As mentioned earlier, vBBU RT threads, namely, ru-thread and lte-softmodem, account for a significant portion of the CPU time allocated for the vBBU. In the configuration with both fixed CPU affinity per thread and CPU affinity to the NIC, as, in IA-TA, RT threads ru-thread and lte-softmodem experience lower scheduling latency compared to IA-WA. However, the impact on non-RT threads is the opposite. In IA-WA, non-RT threads can receive CPU time on any of the CPUs in C_W, resulting in lower scheduling latency for

non-RT threads compared to IA-TA. Consequently, in IA-TA, latency events of non-RT threads even shift beyond the 2–3 microsecond range.

CPU Isolation

Based on the evidence presented thus far, the coexistence of user-space processes and Kernel threads sharing computing resources has a noticeable impact on vBBU performance. In this section, we explore the effectiveness of two CPU isolation mechanisms, namely, Shielding and isolCPU, as potential solutions to mitigate the influence of concurrently executing workloads. Since CPU isolation ensures that all user-space processes run on CPUs other than isolated ones, scenarios such as Idle and User exhibit no direct impact on vBBU performance. Consequently, our evaluation focuses exclusively on the Kernel scenario. To gain deeper insights into the advantages of Shielding compared to isolCPU for the vBBU, we analyzed the performance of vBBU's DL NACK procedures, as illustrated in Figure 7.

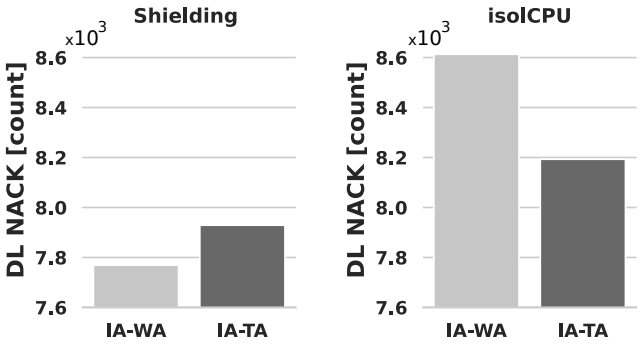


Figure 7. vBBU's number of DL NACKs computed over a set of six experiments (total observation time-span is 60 min). Evaluated scenarios: Kernel–CPU affinity allocation: IA-WA vs. IA-TA.

The Shielding approach demonstrates better performance by producing fewer DL NACKs compared to isolCPU. Remarkably, Shielding outperforms even the results observed in Figure 5 for the Idle scenario, where the vBBU utilizes all available CPUs in the edge server. In this case, the allocation of CPU affinity through IA-WA enhances vBBU performance in terms of DL NACKs, as opposed to IA-TA. In contrast, when employing CPU isolation with isolCPU, the vBBU experiences a higher number of DL NACKs for both IA-WA and IA-TA, surpassing even the performance observed in a stressed system, such as in the User scenario depicted in Figure 5. This discrepancy is attributed to the implications of isolating CPUs in isolCPU, which involves migrating buffered data chunks. Potentially, these chunks may be in the cache of CPUs outside the isolated set, leading to degradation in the DL data path.

Certainly, isolating CPUs mitigates the impact on vBBU scheduling latency caused by sharing computing resources. As shown in Table 5, both CPU isolation approaches result in lower vBBU scheduling latency compared to the results presented in Tables 2–4 for the Kernel scenario.

However, the specific benefit of isolCPU over Shielding in terms of vBBU scheduling latency is not entirely clear. The observed difference in scheduling latency results between the two cases is not significant. Some related research [63] suggests that traffic-based applications benefit from hard isolation, reducing processing latency. However, because the vBBU involves various operations beyond packet processing, such as I/O, such a benefit may not be readily apparent.

While isolating CPUs proves beneficial in mitigating the impact of sharing computing resources, it comes with the trade-off of resource underutilization. Therefore, before adopting this approach, a careful assessment of the trade-off between mitigating processing interference and maintaining resource efficiency is essential.

Table 5. Scheduling latency of vBBU's threads (percentage). Evaluated scenario: Kernel (sharing computing resources with collocated Kernel thread processing generated by incoming packet processing). CPU affinity allocation: IA-WA vs. IA-TA; CPU isolation: Shielding vs isolCPU.

Thread	0–1 (microseconds)				2–3 (microseconds)				4–7 (microseconds)				8–15 (microseconds)			
	Shielding		isolCPU		Shielding		isolCPU		Shielding		isolCPU		Shielding		isolCPU	
	IA-WA	IA-TA	IA-WA	IA-TA	IA-WA	IA-TA	IA-WA	IA-TA	IA-WA	IA-TA	IA-WA	IA-TA	IA-WA	IA-TA	IA-WA	IA-TA
ru-thread	92.8	93.9	92	93.1	0.3	0.3	0.3	0.3	0.2	0.2	0.2	0.3	6.4	5.4	7.1	6.2
lte-softmodem	98.6	98.4	100	98.4	0.5	0.4	0	0.4	0.2	0.2	0	0.2	0.4	0.4	0	0.4
fep_processing	98.1	98.3	98.1	98.5	0.7	0.4	0.7	0.2	0.1	0.1	0.2	0.1	0.8	1	0.9	1
feptx_thread	98.7	99	98.5	99	0.6	0.4	0.9	0.4	0.2	0.1	0.3	0.1	0.2	0.2	0.2	0.2
TASK_GTP	96.8	93.5	96.8	93.5	0.7	4.7	0.7	4.7	1.6	1.2	1.6	1.2	0.6	0.2	0.6	0.2
UDP_TASK	92.6	88.4	92.4	89	1	5.9	1	6	4.8	4.4	4.8	3.7	1.4	0.7	1.3	0.7

5. Assessing Resource Sharing in the Cloud-RAN Architecture

The Cloud-RAN architecture involves multiple vBBUs instantiated on the same edge computer, collectively forming what is known as the vBBU pool. Utilizing virtualization technology, particularly, containers in this study, these vBBUs operate as isolated processes. However, the presence of collocated containers or overloaded Kernel threads, such as those engaged in high-rate packet processing in the Linux Kernel, can have a notable impact on the performance of vBBUs. One contributing factor is the scheduler that allocates CPU time to processes from different containers on the same CPU.

To address this issue, we propose an orthogonal CPU affinity allocation to containers, utilizing the CPU affinity approaches *IA* and *WA*, as introduced in Section 3.

5.1. Mobile Network Scenario

We consider the mobile network scenario illustrated in Figure 8. This configuration features two vBBUs, denoted as vBBU1 and vBBU2, respectively. Each vBBU, deployed with distinct functional splits, operates between the DU and the CU. While vBBU1 and vBBU2 are the only applications instantiated in the DU, they share computing resources at the CU. Moreover, this network scenario involves a single UE connected to each of the vBBUs. The experimental setup for the deployment of this mobile network is summarized in Appendix A.

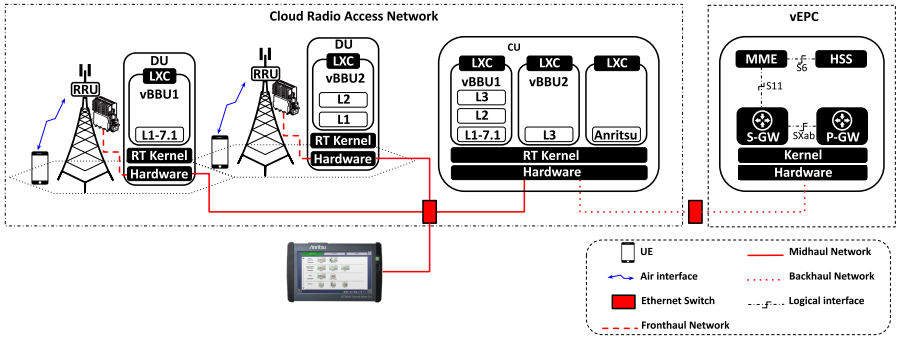


Figure 8. Mobile network scenario based on the Cloud-RAN architecture. Deploying two vBBUs with split 7.1 and split 2, respectively. The mobile transport Xhaul is based on switched Ethernet.

Based on the LTE–BBU functional split model shown in Figure 2, vBBU1 adopts the functional split 7.1, while vBBU2 adopts the functional split 2. Due to split 7.1 hosting most of the BBU RT functions in the CU, vBBU1 imposes greater demands in terms of latency requirements in the Midhaul and processing capacity at the CU. In contrast, vBBU2 deploys all functions involved in the HARQ loop in the DU, making its data-rate requirements in the Midhaul network similar to those in the Backhaul. Here, latency requirements depend on the user's service.

Linux containers (LXC) serve as the virtualization environment for deploying both vBBUs at the DU and CU. To assess resource sharing in Cloud-RAN, we introduce an additional service which involves a traffic flow generated by the Anritsu device and aggregated in the Midhaul. The destination for this traffic is a third LXC is deployed at the CU, referred to as A. Designed as a non-RT user-space application sharing computing and network resources with vBBUs at the CU, A consists of the network utility Netcat, which consumes packets from the traffic flow.

In a containerized virtualization, the NIC is likely shared among different containers. In this network scenario, all services instatiated at the CU—vBBU1, vBBU2, and A—share the NIC port connecting the CU to the Midhaul network. Similarly, vBBU1 and vBBU2 share the second NIC port connecting the CU to the Backhaul. Using two different NIC ports to access the Midhaul and the Backhaul allows for isolating traffic in the Midhaul with time requirements from traffic in the Backhaul, which does not impose any time requirements.

To facilitate NIC sharing among containers, the edge server associated with the CU deploys an SR-IOV-based NIC. This implementation leverages SR-IOV to create multiple virtual functions (VFs) atop the NIC's PCI Express (PCIe) physical function (PF). The NIC's driver supporting SR-IOV registers the corresponding RX/TX Hard-IRQ for that VF. Moreover, a unique MAC address is assigned to each VF. This pair of MAC addresses—RX/TX Hard-IRQ—makes the VF look like an independent NIC itself. Each VF is assigned a unique MAC address and is linked to corresponding RX/TX Hard-IRQs, which together make the VF appear as an independent NIC. Assigned to one of such VF, an LXC can access the network with complete traffic isolation and without relying on any Kernel features for the creation of vNICs assigned to containers or internal virtual switching for their operation. Previous evidence suggests that mechanisms based on creating virtual NIC (vNIC) like macvlan or SR-IOV provide lower overhead than Kernel-based software switch mechanisms like Linux bridge or Open vSwitch (OVS) [76].

As depicted in Figure 9, the NIC used in this network scenario features two Ethernet ports that support SR-IOV (see the hardware details in Table A1). The first port, known as p1, is used in Midhaul. As all three LXCs deployed at the CU are connected to the Midhaul, three VFs are created on top of p1's PF: VF1, VF2, and VF3. Each LXC defines a vNIC called nic1, with a different VF assigned to each LXC, allowing access to the Midhaul. For example, VF1 is assigned to the nic1 of vBBU1, VF2 to the nic1 of vBBU2, and VF3 to the nic1 of A.

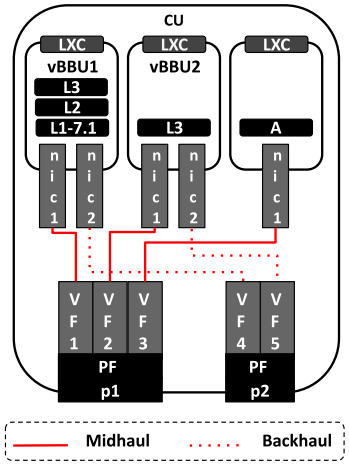


Figure 9. SR-IOV-based NIC sharing in Cloud-RAN-Deploying two different ports: p1 provides access to the Midhaul; p2 provides access to the Backhaul.

On the other hand, the second NIC port, p2, is used in Backhaul. Since only vBBU1 and vBBU2 connect to the Backhaul, two VFs are created on top of p2's PF: VF4 and VF5. Additionally, both vBBU1 and vBBU2 define a second vNIC called nic2. VF4 is assigned to the nic2 of vBBU1, while VF5 is assigned to the nic2 of vBBU2 for access to the Backhaul.

5.2. Sharing Computing Resources at the CU

In the network scenario evaluated in this section, three user-space applications share resources in the CU. Firstly, vBBU1, which deploys split 7.1, is the most time-critical application. Since vBBU1 shares frequency domain samples of the signal between the DU and the CU, it is sensitive to processing interference or resource unavailability. Evidence suggests that running vBBU1 on a minimum of four CPUs guarantees stable performance [50]. On the contrary, vBBU2, which implements split 2, hosts L3 functions at the CU. L3 functions are not time-critical and, therefore, run as non-RT threads in the Linux RT-Kernel. Consequently, in this deployment, two CPUs are assigned to vBBU2. Lastly, service A runs as a best-effort application for BT packets. Thus, no more than a single CPU is assigned to LXC hosting A.

To mitigate processing interference from collocated LXCs, we assess the CPU affinity scenario *IA-WA* introduced in Section 3. Figure 10 shows the CPU affinity assigned to the LXCs hosted at the CU. Following the *WA* approach, while the CPU affinity to vBBU1 is $C_W^{BBU1} = \{C4, C5, C6, C7\}$, the CPU affinity to vBBU2 is defined as $C_W^{BBU2} = \{C2, C3\}$. Similarly, the CPU affinity to A is $C_W^{A} = \{C1\}$. In this way, by pinning LXCs to disjoint subsets of CPUs, user-space process isolation is guaranteed.

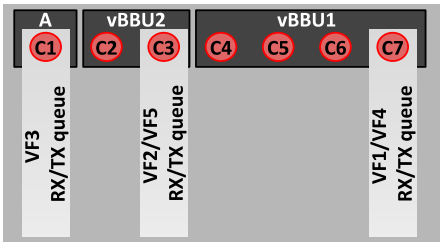


Figure 10. CPU affinity allocation based on *IA-WA* to the vBBU and to the vNIC VF's RX/TX queue Hard-IRQ.

On the other hand, following the *IA* approach, we adopted the CPU affinity to the NIC. This time, the CPU affinities are allocated to the SR-IOV's VFs, as shown in Figure 10. More specifically, the RX/TX Hard-IRQ queues associated with VF1 and VF4 in vBBU1 are allocated CPU affinity $C_{VF1}^{BBU1} = C_{VF4}^{BBU1} = C7$. Similarly, the RX/TX Hard-IRQ queues associated with VF2 and VF5 in vBBU2 are allocated CPU affinity $C_{VF2}^{BBU2} = C_{VF5}^{BBU2} = C3$. Finally, the RX/TX Hard-IRQ queue associated with VF3 in A is allocated CPU affinity $C_{VF3}^{A} = C1$. This way, high-rate incoming packets and their corresponding Hard-IRQs or Soft-IRQs (in NAPI) are processed by a different CPU than the ones used by the vBBUs, thus avoiding processing interference.

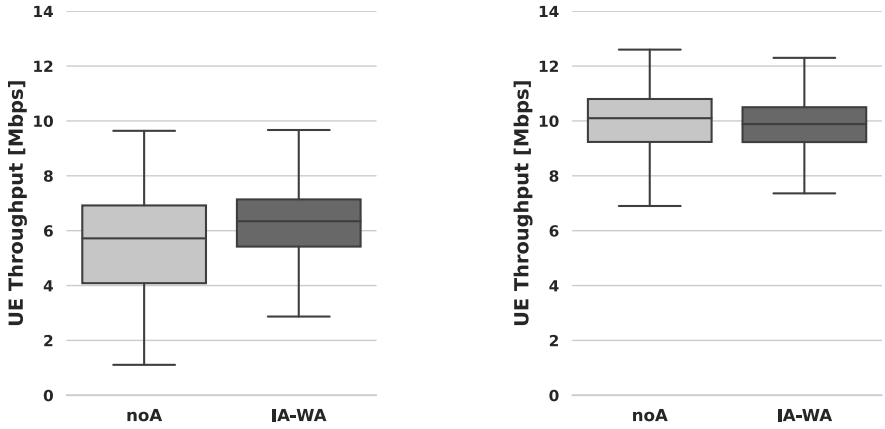
5.3. Results and Discussion

Two scenarios of resource sharing have been considered for evaluation: (i) noA, where no CPU affinity is defined. This scenario provides a baseline for the evaluation, where the RT-Kernel Scheduler decides to which CPUs mapping the resources assigned to a LXC. Such a decision could vary over time depending on the system load, resource availability, and process priorities; (ii) IA-WA, which adopts the CPU affinity approach described above for all LXCs and vNICs. Here, the CPU mapping is predefined according to CPU affinities.

Conducted experiments consist of measuring end-to-end throughput at each UE connected to a given vBBU. In this case, the end-to-end throughput is derived from a TCP downstream flow generated by an Iperf3 server, which is located right outside the mobile network. The destination of the TCP flow is an Iperf3 client instantiated at the UE. A set of six experiments of 10 min each is conducted per UE. In total, two downstream TCP flows, one per UE, are generated with an initial rate of 10 Mbps. While conducting an experiment for a given UE, the TCP downstream flow for the second UE runs as background traffic. In addition, the BT from the Anritsu device to the A service at the CU runs during the whole experimentation. Not only does the BT allow for evaluating traffic aggregation and

network sharing in the Midhaul, but the BT's destination service A also allows evaluating resource sharing at the CU.

Figure 11 shows the results after computing the end-to-end throughput as measured by the UE's Iperf3 client.



(**a**) UE connected to vBBU1-split 7.1 (**b**) UE connected to vBBU2-split 2

Figure 11. Cloud-RAN performance: end-to-end throughput measured by vBBU's UEs. CPU affinity allocation: (i) noA vs. (ii) IA-WA.

The advantages of implementing the proposed CPU affinity are evident in the distribution spread of the data illustrated in Figure 11. Particularly notable is the substantial benefit for the UE connected to vBBU1. In the IA-WA scenario, the median UE's received throughput reaches 6.7 Mbps, outperforming the noA scenario where the median throughput is 5.7 Mbps. Moreover, the Interquartile Range (IQR), representing the range in which 50% of the sample data fall, is reduced from 4.2 in noA to 2.5 in IA-WA. This indicates a 40% reduction in throughput variability achieved by allocating CPU affinity based on IA-WA in comparison with scenarios without CPU affinity.

Although the impact of adopting CPU affinity is not as pronounced for vBBU2 with split 2 as it is for vBBU1, there remains a noticeable difference in the distribution spread compared to the noA scenario. As depicted in Figure 11b, for the UE connected to vBBU2, the IQR is 1.9 in the IA-WA scenario, whereas it is 2.3 in the noA scenario. In other words, by allocating CPU affinity based on IA-WA, the throughput variability is reduced by 21% in comparison with scenarios without CPU affinity.

6. Conclusions

This study provides insights into the impact of resource sharing with collocated workloads on vBBU performance in edge servers managed by the Linux Kernel. The empirical investigation highlights the substantial challenges posed by resource contention when vBBUs share resources with user-space tasks and Kernel threads.

Effective CPU management strategies, particularly, CPU affinity, emerge as crucial mechanisms for significantly enhancing vBBU performance. The application of CPU affinity, demonstrates substantial benefits, including a notable reduction in throughput variability, decreased vBBU NACK ratios, and lowered scheduling latency within the Linux RT-Kernel.

Although CPU isolation exhibits performance comparable to that in a reference scenario where vBBU is the only active process, the trade-off involves the potential underutilization of resources. In contrast, the allocation of orthogonal CPU affinity to containers hosting both vBBUs and best-effort applications demonstrates the potential to guarantee CPU isolation, thereby improving end-to-end vBBU performance in a Cloud-RAN setup.

Looking ahead, the static nature of CPU affinity allocation in this study prompts a call for future exploration into dynamic assignment methodologies. Future research efforts will focus on adapting CPU affinity allocations based on real-time application demands,

CPU usage, and availability, with a view to improving adaptability and responsiveness to varying workloads. Additionally, the research agenda includes the investigation of resource contention scenarios in diverse setups, such as those involving dynamic functional splitting within Cloud-RAN. The ultimate goal is to develop robust strategies and mechanisms that optimize vBBU performance across a spectrum of Cloud-RAN configurations.

While our investigation anchors itself primarily on the exploration of CPU resource sharing and its direct impact on vBBU performance, we recognize the vast landscape of potential performance-affecting factors within a Cloud-RAN environment. Future studies could further this research by considering a more diverse array of performance metrics and external factors that contribute to system efficiency.

As part of our future work, we look forward to comparisons with alternative resource management strategies to ascertain the distinct advantages of CPU affinity and CPU isolation within Cloud-RAN deployments. This study lays the groundwork by establishing a benchmark for the effectiveness of these well-understood strategies in managing vBBU performance in resource-sharing scenarios. We recognize the limitations arising from the scope of experimentation at our disposal and hope that future endeavors could expand upon our work, contributing further to the complex discussion of resource management in real-time systems.

Although the results we have presented provide valuable insights into the performance dynamics of vBBUs within our experimental configuration, we fully recognize that these findings are contingent on the specific conditions and constraints of our setup. These results serve as a snapshot of performance under certain hardware specifications, network conditions, and software settings, providing a focused examination that is inherently limited in scope. Consequently, the broader applicability of our findings to different configurations warrants caution and further investigation. Moving forward, future research could benefit from a diverse range of experimental scenarios, including varying hardware platforms and network loads, to enhance the generalizability of results and provide a more versatile understanding of CPU management strategies in varied Cloud-RAN environments.

While our study successfully establishes the impact of resource sharing on vBBU performance and illustrates the utility of CPU affinity and isolation as mitigation strategies, we acknowledge that transitioning these strategies to a large-scale Cloud-RAN deployment presents its own set of challenges. The scalability of these CPU management techniques and their adaptability within a dynamic and heterogeneous network environment remain crucial factors for real-world implementation. Future work could specifically focus on the practical aspects of deploying these strategies, with an emphasis on automation, scalability, and management within broader Cloud-RAN frameworks.

Author Contributions: Conceptualization, A.F.O., M.-R.F., A.E. and H.B.; methodology, A.F.O., M.-R.F., A.E. and H.B.; software, A.F.O.; validation, A.F.O., M.-R.F. and A.E.; formal analysis, A.F.O., M.-R.F. and A.E.; investigation, A.F.O., M.-R.F., A.E. and H.B.; resources, A.F.O. and H.B.; data curation, A.F.O.; writing—original draft preparation, A.F.O.; writing—review and editing, A.F.O., M.-R.F. and A.E.; visualization, A.F.O.; supervision, A.E. and H.B.; project administration, A.E. and H.B.; funding acquisition, A.E. and H.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: Author Ahmed Elmokashfi was employed by the company Amazon Web Services (AWS). The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Appendix A. Testbed Setup Specifications

Table A1 provides an overview of the hardware and software specifications for deploying the experimental setup corresponding to the mobile network scenarios depicted in Figures 3 and 8.

Table A1. Testbed setup for the mobile network scenario corresponding to both the monolithic vBBU deployment in Figure 3 and the Cloud-RAN deployment in Figure 8—hardware and software specifications.

Component	Description
UE	OnePlus-5 phone/Software: OxygenOS Version 10.0.1/Oneplus, Shenzhen, China.
air interface (as specified by the vBBU)	25 Physical Resource Blocks (PRB), which provides 5 MHz bandwidth.
RRU	Ettus (B210) One antenna port—Single Input Single Output (SISO)/Software: Universal Software Radio Peripheral (USRP) Version: 4.6.0.0-7-gece7c4811/Ettus Research, Austin, TX, USA.
Fronthaul	Fast SuperSpeed USB 3.0 connectivity at 5.0 Gbit/s (provided by Ettus (B210)).
	Monolithic vBBU deployment (see Figure 3)
vBBU	Monolithic eNodeB-LTE implementation from OpenAirInterface (OAI) Version 2.0.0.
Edge server	Equipped with eight Intel i7-8750H physical processors at 2.20 GHz, and 32 GiB of memory.
Edge server's OS	Ubuntu 18.04 with low-latency Linux kernel version 5.3.28.
Edge server's NIC	Supermicro AOC-SG-i2 Gigabit Ethernet adapter, equipped with two Intel 82575 Gigabit Ethernet ports/San Jose, CA, USA.
Backhaul	physical link at 1 Gbit/s.
	Cloud-RAN (see Figure 8)
vBBU1	OpenAirInterface eNodeB–LTE implementation Version 2.0.0, with split 7.1. This functional split uses the NGFI-IF4p5 interface specification [77]
vBBU2	OpenAirInterface eNodeB–LTE implementation Version 2.0.0, with split 2 using the F1 Application Protocol (F1AP) [78,79].
DU	Intel NUC7i7BNB equipped with four Intel Core i7-7567U processors, and 32 GiB of memory.
DU's OS	Ubuntu 16.04 with low-latency Linux kernel version 4.19.58.
CU	Edge server equipped with eight Intel i7-8750H physical processors at 2.20 GHz, and 32 GiB of memory.
CU's OS	Ubuntu 20.04 with Linux Kernel low-latency patch version 5.4.0.125.126.
Midhaul & Backhaul	Juniper EX4200 Ethernet switch, with physical interfaces at 1 Gbit/ Software: Junos OS Version 12.3R12-S21.
	Core Network
vEPC	4G EPC implementation from OAI.
vEPC's host physical machine	Generic server equipped with four Intel i7 processor at 2.20GHz, and 12 GiB of memory.
vEPC's host OS	Ubuntu 18.04 with generic Linux kernel version 5.3.28.

References

1. Checko, A.; Christiansen, H.L.; Yan, Y.; Scolari, L.; Kardaras, G.; Berger, M.S.; Dittmann, L. Cloud RAN for Mobile Networks—A Technology Overview. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 405–426. [CrossRef]
2. Mosnier, A. Embedded/Real-Time Linux Survey. Available online: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0c81e1915ba80e319739b988ee74a31ef853c7b6 (accessed on 20 December 2023).
3. Karachatzis, P.; Ruh, J.; Craciunas, S.S. An Evaluation of Time-Triggered Scheduling in the Linux Kernel. In Proceedings of the RTNS '23: 31st International Conference on Real-Time Networks and Systems, New York, NY, USA, 7–8 June 2023; pp. 119–131. [CrossRef]
4. Yodaiken, V. The rtlinux manifesto. In Proceedings of the 5th Linux Expo, Raleigh, NC, USA, 18–22 May 1999.
5. Ubuntu. Linux Low Latency. Available online: https://packages.ubuntu.com/search?keywords=linux-lowlatency (accessed on 5 April 2024).
6. Foundation, T.L. PREEMPT_RT Patch. Available online: https://wiki.linuxfoundation.org/realtime/start (accessed on 5 April 2024).
7. Nikaein, N.; Knopp, R.; Kaltenberger, F.; Gauthier, L.; Bonnet, C.; Nussbaum, D.; Ghaddab, R. OpenAirInterface: an open LTE network in a PC. In Proceedings of the 20th Annual International Conference on Mobile Computing and Networking, Maui, HI, USA, 7–11 September 2014; pp. 305–308.

8. Harutyunyan, D.; Riggio, R. Flex5G: Flexible Functional Split in 5G Networks. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 961–975. [CrossRef]
9. Alba, A.M.; Velásquez, J.H.G.; Kellerer, W. An adaptive functional split in 5G networks. In Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Paris, France, 29 April–2 May 2019; pp. 410–416. [CrossRef]
10. Azariah, W.; Bimo, F.A.; Lin, C.W.; Cheng, R.G.; Jana, R.; Nikaein, N. A survey on open radio access networks: Challenges, research directions, and open source approaches. *arXiv* **2022**, arXiv:2208.09125.
11. Abeni, L.; Cucinotta, T.; Pinczel, B.; Mátray, P.; Srinivasan, M.K.; Lindquist, T. On the Use of Linux Real-Time Features for RAN Packet Processing in Cloud Environments. In *Proceedings of the High Performance Computing. ISC High Performance 2022 International Workshops*; Anzt, H., Bienz, A., Luszczek, P., Baboulin, M., Eds.; Springer: Cham, Switzerland, 2022; pp. 371–382.
12. Xu, C.; Zhao, Z.; Wang, H.; Shea, R.; Liu, J. Energy Efficiency of Cloud Virtual Machines: From Traffic Pattern and CPU Affinity Perspectives. *IEEE Syst. J.* **2017**, *11*, 835–845. [CrossRef]
13. Iorgulescu, C.; Azimi, R.; Kwon, Y.; Elnikety, S.; Syamala, M.; Narasayya, V.; Herodotou, H.; Tomita, P.; Chen, A.; Zhang, J.; et al. PerfIso: Performance Isolation for Commercial Latency-Sensitive Services. In Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC 18), Boston, MA, USA, 11–13 July 2018; pp. 519–532.
14. Cinque, M.; Cotroneo, D.; De Simone, L.; Rosiello, S. Virtualizing mixed-criticality systems: A survey on industrial trends and issues. *Future Gener. Comput. Syst.* **2022**, *129*, 315–330. [CrossRef]
15. Åsberg, M.; Nolte, T.; Kato, S.; Rajkumar, R. ExSched: An External CPU Scheduler Framework for Real-Time Systems. In Proceedings of the 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Seoul, Republic of Korea, 19–22 August 2012; pp. 240–249. [CrossRef]
16. Valsan, P.K.; Yun, H.; Farshchi, F. Taming Non-Blocking Caches to Improve Isolation in Multicore Real-Time Systems. In Proceedings of the 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, 11–14 April 2016; pp. 1–12. [CrossRef]
17. Ruiz, A.P.; Rivas, M.A.; Harbour, M.G. CPU Isolation on the Android OS for Running Real-Time Applications. In Proceedings of the JTRES '15: 13th International Workshop on Java Technologies for Real-Time and Embedded Systems, New York, NY, USA, 7–8 October 2015. [CrossRef]
18. Angui, B.; Corbel, R.; Rodriguez, V.Q.; Stephan, E. Towards 6G zero touch networks: The case of automated Cloud-RAN deployments. In Proceedings of the 2022 IEEE 19th Annual Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2022; pp. 1–6. [CrossRef]
19. Habibi, M.A.; Han, B.; Nasimi, M.; Kuruvatti, N.P.; Fellan, A.; Schotten, H.D., Towards a Fully Virtualized, Cloudified, and Slicing-Aware RAN for 6G Mobile Networks. In *6G Mobile Wireless Networks*; Wu, Y., Singh, S., Taleb, T., Roy, A., Dhillon, H.S., Kanagarathinam, M.R.; De, A., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 327–358. [CrossRef]
20. 3GPP TR 38.801, Study on New Radio Access Technology: Radio Access Architecture and Interfaces. 2017. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3056 (accessed on 5 April 2024).
21. Larsen, L.M.P.; Checko, A.; Christiansen, H.L. A Survey of the Functional Splits Proposed for 5G Mobile Crosshaul Networks. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 146–172. [CrossRef]
22. *IEEE Std 1914.1-2019*; Standard for Packet-based Fronthaul Transport Network. IEEE: Piscataway, NJ, USA, 2019.
23. Joda, R.; Pamuklu, T.; Iturria-Rivera, P.E.; Erol-Kantarci, M. Deep Reinforcement Learning-Based Joint User Association and CU–DU Placement in O-RAN. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 4097–4110. [CrossRef]
24. Tomaszewski, L.; Kukliński, S.; Kołakowski, R. A New Approach to 5G and MEC Integration. In *Proceedings of the Artificial Intelligence Applications and Innovations. AIAI 2020 IFIP WG 12.5 International Workshops*; Maglogiannis, I., Iliadis, L., Pimenidis, E., Eds.; Springer: Cham, Switzerland, 2020; pp. 15–24.
25. Reghenzani, F.; Massari, G.; Fornaciari, W. The Real-Time Linux Kernel: A Survey on PREEMPT_RT. *ACM Comput. Surv.* **2019**, *52*, 36. [CrossRef]
26. RTAI. RTAI-the Real-Time Application Interface for Linux. Available online: https://www.rtai.org/ (accessed on 20 December 2023).
27. Kaltenberger, F.; Wagner, S. Experimental analysis of network-aided interference-aware receiver for LTE MU-MIMO. In Proceedings of the 2014 IEEE 8th Sensor Array and Multichannel Signal Processing Workshop (SAM), Coruna, Spain, 22–25 June 2014; pp. 325–328. [CrossRef]
28. Alyafawi, I.; Schiller, E.; Braun, T.; Dimitrova, D.; Gomes, A.; Nikaein, N. Critical issues of centralized and cloudified LTE-FDD radio access networks. In Proceedings of the 2015 IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; pp. 5523–5528.
29. Bhaumik, S.; Chandrabose, S.P.; Jataprolu, M.K.; Kumar, G.; Muralidhar, A.; Polakos, P.; Srinivasan, V.; Woo, T. CloudIQ: A framework for processing base stations in a data center. In Proceedings of the 18th Annual International Conference on Mobile Computing and Networking, Istanbul, Turkey, 22–26 August 2012; pp. 125–136.
30. Fajjari, I.; Aitsaadi, N.; Amanou, S. Optimized Resource Allocation and RRH Attachment in Experimental SDN based Cloud-RAN. In Proceedings of the 2019 16th IEEE Annual Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 11–14 January 2019; pp. 1–6.

31. Younis, A.; Tran, T.X.; Pompili, D. Bandwidth and Energy-Aware Resource Allocation for Cloud Radio Access Networks. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 6487–6500. [CrossRef]
32. Nikaein, N. Processing Radio Access Network Functions in the Cloud: Critical Issues and Modeling. In Proceedings of the MCS '15: 6th International Workshop on Mobile Cloud Computing and Services, New York, NY, USA, 11 September 2015; pp. 36–43. [CrossRef]
33. Huang, S.; Luo, Y.; Chen, B.; Chung, Y.; Chou, J. Application-Aware Traffic Redirection: A Mobile Edge Computing Implementation Toward Future 5G Networks. In Proceedings of the 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2), Los Alamitos, CA, USA, 22–25 November 2017; pp. 17–23. [CrossRef]
34. Xi, S.; Xu, M.; Lu, C.; Phan, L.T.X.; Gill, C.; Sokolsky, O.; Lee, I. Real-time multi-core virtual machine scheduling in Xen. In Proceedings of the 2014 International Conference on Embedded Software (EMSOFT), New Delhi, India, 12–17 October 2014; pp. 1–10. [CrossRef]
35. Struhár, V.; Behnam, M.; Ashjaei, M.; Papadopoulos, A.V. Real-Time Containers: A Survey. In *Proceedings of the 2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020)*; Cervin, A., Yang, Y., Eds.; OpenAccess Series in Informatics (OASIcs): Dagstuhl, Germany, 2020; Volume 80, pp. 7:1–7:9. [CrossRef]
36. Abeni, L.; Balsini, A.; Cucinotta, T. Container-Based Real-Time Scheduling in the Linux Kernel. *SIGBED Rev.* **2019**, *16*, 33–38. [CrossRef]
37. Nikaein, N.; Schiller, E.; Favraud, R.; Knopp, R.; Alyafawi, I.; Braun, T., Towards a Cloud-Native Radio Access Network. In *Advances in Mobile Cloud Computing and Big Data in the 5G Era*; Mavromoustakis, C.X., Mastorakis, G., Dobre, C., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 171–202. [CrossRef]
38. Mao, C.N.; Huang, M.H.; Padhy, S.; Wang, S.T.; Chung, W.C.; Chung, Y.C.; Hsu, C.H. Minimizing Latency of Real-Time Container Cloud for Software Radio Access Networks. In Proceedings of the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, Canada, November 30–3 December 2015; pp. 611–616. [CrossRef]
39. Cinque, M.; De Tommasi, G. Work-in-Progress: Real-Time Containers for Large-Scale Mixed-Criticality Systems. In Proceedings of the 2017 IEEE Real-Time Systems Symposium (RTSS), Paris, France, 5–8 December 2017; pp. 369–371. [CrossRef]
40. Han, W.T.; Knopp, R. OpenAirInterface: A Pipeline Structure for 5G. In Proceedings of the 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), Shanghai, China, 19–21 November 2018; pp. 1–4. [CrossRef]
41. Foukas, X.; Nikaein, N.; Kassem, M.M.; Marina, M.K.; Kontovasilis, K. FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks. In Proceedings of the CoNEXT '16: 12th International on Conference on Emerging Networking EXperiments and Technologies, New York, NY, USA, 12–15 December 2016; pp. 427–441. [CrossRef]
42. Cavicchioli, R.; Capodieci, N.; Bertogna, M. Memory interference characterization between CPU cores and integrated GPUs in mixed-criticality platforms. In Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 12–15 September 2017; pp. 1–10. [CrossRef]
43. Burns, A.; Davis, R.I. A Survey of Research into Mixed Criticality Systems. *ACM Comput. Surv.* **2017**, *50*, 1–37. [CrossRef]
44. De, P.; Mann, V.; Mittaly, U. Handling OS jitter on multicore multithreaded systems. In Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing, Chengdu, China, 10–12 August 2009; pp. 1–12. [CrossRef]
45. Reghenzani, F.; Massari, G.; Fornaciari, W. Mixed Time-Criticality Process Interferences Characterization on a Multicore Linux System. In Proceedings of the 2017 Euromicro Conference on Digital System Design (DSD), Vienna, Austria, 30 August–1 September 2017; pp. 427–434. [CrossRef]
46. Barletta, M.; Cinque, M.; De Simone, L.; Della Corte, R. Achieving isolation in mixed-criticality industrial edge systems with real-time containers. In Proceedings of the 34th Euromicro Conference on Real-Time Systems (ECRTS 2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Modena, Italy, 5–8 July 2022.
47. Burns, A.; Davis, R.I. Mixed Criticality Systems—A Review. Available online: https://www-users.york.ac.uk/~ab38/review.pdf (accessed on 20 December 2023).
48. Liu, L.; Wang, H.; Wang, A.; Xiao, M.; Cheng, Y.; Chen, S. Mind the Gap: Broken Promises of CPU Reservations in Containerized Multi-Tenant Clouds. In Proceedings of the SoCC '21: ACM Symposium on Cloud Computing, New York, NY, USA, 1–4 November 2021; pp. 243–257. [CrossRef]
49. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [CrossRef]
50. Foukas, X.; Radunovic, B. Concordia: Teaching the 5G VRAN to Share Compute. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference, New York, NY, USA, 23–27 August 2021; SIGCOMM '21, pp. 580–596. [CrossRef]
51. Nikaein, N.; Marina, M.K.; Manickam, S.; Dawson, A.; Knopp, R.; Bonnet, C. OpenAirInterface: A Flexible Platform for 5G Research. *SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 33–38. [CrossRef]
52. Gomez-Miguelez, I.; Garcia-Saavedra, A.; Sutton, P.D.; Serrano, P.; Cano, C.; Leith, D.J. In Proceedings of the SrsLTE: An Open-Source Platform for LTE Evolution and Experimentation, New York, NY, USA, 3–7 October 2016; WiNTECH '16, pp. 25–32. [CrossRef]
53. De Oliveira, D.B.; De Oliveira, R.S. Timing analysis of the PREEMPT RT Linux kernel. *Software Pract. Exp.* **2016**, *46*, 789–819. [CrossRef]
54. Abeni, L.; Kiraly, C. Investigating the network performance of a real-time Linux Kernel. In Proceedings of 15th Real Time Linux Workshop (RTLWS 2013), Lugano, Switzerland, 28–31 October 2013.

55. Emmerich, P.; Raumer, D.; Beifuß, A.; Erlacher, L.; Wohlfart, F.; Runge, T.M.; Gallenmüller, S.; Carle, G. Optimizing latency and CPU load in packet processing systems. In Proceedings of the 2015 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Chicago, IL, USA, 26–29 July 2015; pp. 1–8. [CrossRef]
56. Wu, W.; Crawford, M.; Bowden, M. The performance analysis of linux networking—Packet receiving. *Comput. Commun.* **2007**, *30*, 1044–1057. [CrossRef]
57. Beifuß, A.; Raumer, D.; Emmerich, P.; Runge, T.M.; Wohlfart, F.; Wolfinger, B.E.; Carle, G. A study of networking software induced latency. In Proceedings of the 2015 International Conference and Workshops on Networked Systems (NetSys), Agadir, Morocco, 13–15 May 2015; pp. 1–8. [CrossRef]
58. Love, R. Kernel Korner: CPU Affinity. *Linux J.* **2003**, *2003*, 8.
59. Ribeiro, C.P.; Castro, M.; Marangonzova-Martin, V.; Méhaut, J.F.; De Freitas, H.C.; da Silva Martins, C.A.P. Evaluating CPU and memory affinity for numerical scientific multithreaded benchmarks on multi-cores. *IADIS Int. J. Comput. Sci. Inf. Syst. (IJCSIS)* **2012**, *1*, 79–93.
60. Kafshdooz, M.M.; Taram, M.; Assadi, S.; Ejlali, A. A compile-time optimization method for WCET reduction in real-time embedded systems through block formation. *ACM Trans. Archit. Code Optim. (TACO)* **2016**, *12*, 1–25. [CrossRef]
61. Ghatrehsamani, D.; Denninnart, C.; Bacik, J.; Amini Salehi, M. The Art of CPU-Pinning: Evaluating and Improving the Performance of Virtualization and Containerization Platforms. In Proceedings of the ICPP '20: 49th International Conference on Parallel Processing-ICPP, New York, NY, USA, 17–19 August 2020. [CrossRef]
62. Bharti, C.; Kanagarathinam, M.R.; Srivastava, S.K.; Lee, M.; Han, J.; Oh, W. CAA: CLAT Aware Affinity Scheduler for Next Generation Mobile Networks. In Proceedings of the 2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC), Las Vegas, NV, USA, 11–14 January 2019; pp. 1–6. [CrossRef]
63. Gutiérrez, C.S.V.; Juan, L.U.S.; Ugarte, I.Z.; Vilches, V.M. Real-time Linux communications: an evaluation of the Linux communication stack for real-time robotic applications. *arXiv* **2018**, arXiv:1808.10821.
64. Derr, S. CPUSETS. Available online: https://www.kernel.org/doc/Documentation/cgroup-v1/cpusets.txt (accessed on 21 March 2023).
65. Delgado, R.; Choi, B.W. New Insights Into the Real-Time Performance of a Multicore Processor. *IEEE Access* **2020**, *8*, 186199–186211. [CrossRef]
66. Kroah-Hartman, G. *Linux Kernel in a Nutshell: A Desktop Quick Reference*; O'Reilly Media, Inc.: Newton, MA, USA, 2006.
67. Kim, S.; Park, H.S. A Multi-core Based Real-time Scheduler Supporting Periodic and Sporadic Threads and Processes. *Int. J. Control. Autom. Syst.* **2023**, *21*, 3048–3056. [CrossRef]
68. Mortimer, M. iperf3 Documentation. Available online: https://buildmedia.readthedocs.org/media/pdf/iperf3-python/latest/iperf3-python.pdf (accessed on 21 November 2023).
69. King, C. Stress-ng. Available online: https://wiki.ubuntu.com/Kernel/Reference/stress-ng (accessed on 21 November 2023).
70. Anritsu. Anritsu MT1000A Network Master Pro Tester. Available online: https://www.anritsu.com/en-us/test-measurement/products/mt1000a (accessed on 21 November 2023).
71. Giacobbi, G. The GNU Netcat Project. Available online: http://netcat.sourceforge.net (accessed on 21 November 2023).
72. 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) Protocol Specification. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2437 (accessed on 21 November 2023).
73. IO-Visor. BPF Compiler Collection (BCC). Available online: https://github.com/iovisor/bcc (accessed on 21 November 2023).
74. Krzywinski, M.; Altman, N. Visualizing samples with box plots. *Nat. Methods* **2014**, *11*, 119–121. [CrossRef] [PubMed]
75. Ho, A.; Smith, S.; Hand, S. On deadlock, livelock, and forward progress. In *Technical Report UCAM-CL-TR-633*; University of Cambridge, Computer Laboratory: Cambridge, UK, 2005. [CrossRef]
76. Claassen, J.; Koning, R.; Grosso, P. Linux containers networking: Performance and scalability of kernel modules. In Proceedings of the NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, Turkey, 25–29 April 2016; pp. 713–717.
77. Knopp, R. Overview of Functional Splits in OAI. Available online: https://www.openairinterface.org/docs/workshop/5_OAI_Workshop_20180620/KNOPP_OAI-functional-splits.pdf (accessed on 21 November 2023).
78. 3GPP. 5G; NG-RAN; F1 Application Protocol (F1AP) (3GPP TS 38.473 Version 15.2.1 Release 15). Available online: https://www.etsi.org/deliver/etsi_ts/138400_138499/138473/15.02.01_60/ts_138473v150201p.pdf (accessed on 21 November 2023).
79. OAI. F1 Interface. Available online: https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/f1-interface (accessed on 20 December 2022).