Containing Low Tail-Latencies in Packet Processing Using Lightweight Virtualization

Florian Wiedner, Max Helm, Alexander Daichendt, Jonas Andre, Georg Carle
Department of Computer Engineering
Technical University of Munich
Garching by Munich, Germany
{wiedner, helm, daichend, andre, carle}@net.in.tum.de

***Abstract*—Packet processing in current network scenarios faces complex challenges due to the increasing prevalence of requirements such as low latency, high reliability, and resource sharing. Virtualization is a potential solution to mitigate these challenges by enabling resource sharing and on-demand provisioning; however, ensuring high reliability and ultra-low latency remains a key challenge. Since bare-metal systems are often impractical because of high cost and space usage, and virtual machines require substantial additional resources, we evaluate the utilization of containers as a potential lightweight solution for low-latency-enabled packet processing. Herein, we discuss the benefits and drawbacks and encourage the use of container environments in low-latency packet processing when the degree of isolation of customer data is adequate and bare metal systems are unaffordable. Our results demonstrate that containers achieve similar latency performance with more predictable tail-latency behavior compared to bare metal packet processing. Furthermore, we show that the overhead caused by virtualization is negligible in tail latencies.**

***Index Terms*—low-latency, testbed, container, virtualization, packet processing**

I. Introduction

Autonomous driving and remote medical procedures are examples of applications that require low-latency network communications and commonly operate on specialized hardware machines to meet this requirement. However, this use of specialized machines creates challenges in scalability, which affects the cost per service rate when real-time requirements are in play. The 5G ultra-reliable low-latency communications (URLLC) profile provides a framework for such systems that require ultra-low latency, defined as $<1\,\mathrm{ms}$ end-to-end latency and 99.999^{th} percentile of traffic must be within this limit [1]. Using dedicated hardware to run applications for such purposes is not an efficient solution economically for the customer or provider.

Therefore, a solution that offers on-demand provisioning and resource sharing for low-latency network services is required. Virtualization of computer systems is one such solution. However, the use of virtual machines (VMs) with a complete operating system (OS) results in significant overhead in terms of performance, memory, and disk space usage. Gallenmüller et al. [2] compared packet processing between bare metal and VMs on commodity hardware and reported that tuning Linux to reduce interrupts and other influences significantly reduce tail latency in packet processing. VMs offer a high level of isolation that is unnecessary in many cases, hence a lighter version is preferred for improving resource-usage.

Containers offer a lightweight virtualization alternative for resource sharing with other containers and host OS on the same system. While containers do not virtualize the complete OS, there are several lightweight software isolation mechanisms available [3].

Given the significance of low-latency in critical systems, evaluating the tail-latency behavior of packet processing during the long-term execution of applications in containers is essential. Herein, we provide

1) a tool for external control of Linux containers (LXC) in experiment automation,
2) an investigation of the influence of optimization techniques on tail-latency,
3) a model for tail-latency behavior in packet processing within containers, and
4) comparison of using network packet processing applications on bare metal, in containers, and in VMs for low-latency optimized, commodity off-the-shelf systems for the use with URLLC.

The paper is structured as follows: Section II offers background information and presents the current development and research progress in virtualization and latency optimizations. In Section III, the challenges involved in low-latency packet processing using containers are discussed. Section IV outlines optimization techniques for containers. Section V describes the measurement setup and Section VI evaluates the proposed approach's results. We provide models of the tail latencies in Section VII. Section VIII offers recommendations on the usage of the specific virtualization technique in low-latency packet processing. Sections IX to XI conclude the paper by presenting limitations, reproducibility information, a conclusion, and future scope for research and development.

II. Background and Related Work

This section presents an analysis of relevant literature in the fields of containers, VMs, remote device management, low-latency applications and optimization, and tail-latency models.

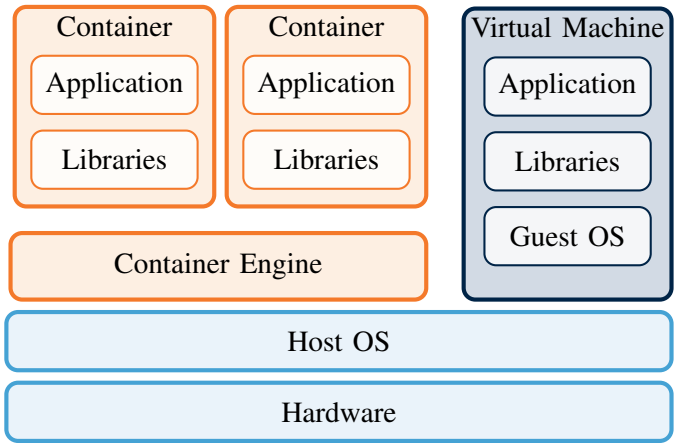

Figure 1. Comparison between container (left) and VMs (right) [5]

A. Containers and VMs

Using a single hardware machine for each customer or application is neither cost-effective nor flexible. Therefore, virtualization is a crucial technology that enables resource sharing and flexible, on-demand provisioning of resources. However, when executed in a virtual environment, applications with strict low-latency and reliability requirements should perform similar to those implemented on bare metal.

Two commonly used architectures for virtualization are hypervisor- and container-based. Hypervisor-based virtualizations (VMs) isolate the complete OS, including the kernel. Whereas, containers are called lightweight or OS-level virtualizations with having the kernel shared between host OS and containers [4]. Containers isolate mainly processes, files, and resource access [4]. In Figure 1, the base shows the hardware running the host OS and the top shows the types of virtualization that are available; the left side depicts containerization, which includes the container engine used to manage the containers; and the right side depicts a VM, which shows the additional overhead of the guest OS residing within each system.

Yadav et al. [5] describe that VMs offer a strict separation using virtualized hardware, and a completely separate OS providing a high level of isolation and reducing the influence of customers on each other on the same physical machine. However, this isolation level results in a significant overhead in resource usage, making VMs ideal for experimental and high-security applications, with a trade-off between security and resources important for URLLC applications [6].

Yadav et al. [5] describe containerization as less resource intensive and highly flexible compared to VMs. With a shared kernel, container offer quick startup and direct device access as Gedia and Perigo [7] have demonstrated. This minimized overhead makes containers an ideal choice for performance critical scenarios where multiple applications must interact with each other [8]. Linux offers several container frameworks, such as Kubernetes, a cluster manager, which automates deployment and enhances portability for applications, or LXC. LXC integrates all libraries of a complete OS, but uses a more complex setup compared to solutions like Docker, and has less overhead [8].

Throughput analysis of containers and VMs is a common area of research, evident from the considerable attention given to understand throughput boundaries. Barham et al. [9] studied the impact of CPU resources on XEN-based VMs focusing on variations induced by time slices on the CPU. Furthermore, Abeni et al. [10] analyzed the effect of tuning Linux on the maximum packet rate of kernel virtual machine (KVMs) and achieved promising results by binding CPU affinity of interrupts to selected cores and the VMs to remaining cores. Similarly, Tran and Kim [11] found that CPU core assignment for containers is crucial for improving throughput. Morabito et al. [12] conclude that containers challenge traditional systems in terms of resource usage and performance. Furthermore, Cha and Kim [13] employed containers to offer low-latency edge services and demonstrated that container setups achieve near-optimal throughput by utilizing hardware support.To conclude, research on packet processing in virtualized systems primarily focuses on throughput analysis or overlay networks such as [14], [15].

Several studies have analyzed latency on VMs as demonstrated in [2], [16], [17], and [18]. However, analyzing latency for packet processing applications based on containers is typically not addressed in present studies, despite the necessity to examine influences on packet processing through containers.

B. Remote Management of Devices

Remote hardware control is a common practice in commercial systems to perform tasks, such as power cycling, or provisioning, remotely. A common protocol to control hardware interfaces is the Intelligent Platform Management Interface (IPMI) [19] using the Baseboard Management Controller (BMC) to control hardware features. Despite the lack of recent specification updates, IPMI is still a widely used protocol as changing the hardware is impractical.

Virtualization solutions are typically controlled using proprietary solutions offering extensive, and exclusive features. Combining hardware machines, VMs, and containers in one ecosystem requires a common interface for all three. VMs and hardware can be controlled using IPMI with utilizing VirtualBMC for VMs [20]. However, for container solutions, such a system does not exists currently. Therefore, an interface supporting IPMI for containers is essential improving the maintainability of, for example, data centers additionally.

C. Low-latency Applications

Packet-processing applications with end-to-end latency requirements of $<1\,\mathrm{ms}$ in general traffic networks are becoming increasingly important. Therefore, packet processing applications must improve latency and reliability.

Gallenmüller et al. [21] analyzed latency implications on intrusion detection systems and found that using a specific OS, reducing interrupts, and using a specific network interface card (NIC) help to reduce latency spikes. As Bozilov et

al. [22] reported, is adding important security introducing additional latency. Therefore, it is crucial to reduce network-induced latency to allow security mechanisms within networks. Jain et al. [23] have shown that improving the data plane on specialized systems can significantly reduce latency.

Other examples of low-latency applications include data center internal communication [24], communication phases in distributed machine learning applications [25], and cloud systems providing centralized services to multiple users as outlined by Gandhi et al. [26].

D. Low-latency OS Optimizations

Tuning and optimizing OS is essential to enable predictable, reliable, and low-latency applications on any system type, whether container, VM, or bare metal. Previous studies report that tunings in specific areas are possible, such as reducing the impact of IO processing on container, and reducing the influences of the system itself. For instance, Gallenmüller et al. [21] have demonstrated that interrupts on the packet-processing core have a significant impact on latency. Additionally, disabling timer ticks, isolating cores, and reducing energy-saving mechanisms on VMs can minimize the impact on packet processing through virtualization [6]. Using poll mode instead of interrupt-based drivers improves latency and reduces the number of context switches. Handley et al. [27] found that using the data plane development kit (DPDK) [28], a framework for poll-mode networking, significantly reduces processing latency as the application can be isolated from the OS kernel.

Similar to VMs, container performance can be improved by reducing interrupts, and adding predictability [29]. Herein, we evaluate and demonstrate the optimization potential for containers in comparison to packet processing on VMs or bare metal including the exploration of further optimizations for low-latency-networking.

E. Tail-latency Models

Extreme Value Theory (EVT) is a statistical technique used to model the behavior of distribution tails [30]. It relies on historical data to predict the probability and magnitude of rare events, such as natural disasters. In the networking domain, EVT can be utilized to model tail-latencies since they constitute similarly rare events. Previous studies have successfully applied EVT to analyze time sequences in both wired and wireless networks [31]–[34]. Our study adopts a similar approach to predict and validate the likelihood of latency spikes. The validation shows an accurate predictive power of such models when extrapolating tail-latency magnitudes and frequencies to longer measurement periods. Additionally, we evaluate the convergence of these predictions and compare different container optimization techniques.

III. Container and Low-latency: Challenges

Using containers for low-latency and highly reliable systems can be challenging but feasible. The processing time on the container node is prolonged due to interrupts needed for the container engine. To mitigate this issue, one method is the use of poll-mode-drivers that minimize context switches and interrupts. DPDK [28] provides user-space, poll-mode networking to accelerate packet processing, with a broad range of available applications such as MoonGen [35], a high-speed packet generator. Using user-space networking within containers requires additional tasks performed outside of the container as not all operations are allowed within [11].

Table I
Latency optimized boot parameters for Host OS running containerized systems

Parameter	Value	Description
rcu_nocbs	[*cores*]	No RCU callbacks
rcu_nocbs_poll		No RCU callback threads wakeup
irqaffinity	0	Interrupts on specific core
idle	poll	Poll mode when core idle
tsc	reliable	Rely on TSC without check
mce	ignore_ce	Ignore corrected errors
audit	0	Disable audit messages
nmi_watchdog	0	Disable NMI watchdog
skew_tick	1	No simultaneous ticks for locks
nosoftlookup		Disables logging of backtraces
nosmt		Disables hyperthreading

Moreover, containers cannot be entirely isolated from the host OS as a shared kernel is used, and interrupts are needed on the specific cores, resulting in the challenge of optimizing them for low-latency operations with reducing the influence of operations performed outside of the containers. The challenges presented herein illustrate the trade-off between resource sharing and URLLC.

IV. Optimization Analysis

In computer systems, processes can be affected by interrupts, the sleep state of CPUs, or concurrent processes. While throughput in packet processing is unaffected by these influences, latency, specifically tail-latency, is significantly impacted. Having examined optimizations in both bare-metal and VM environments, we evaluate the suitability of these optimizations for container environments.

Several studies have examined network latency optimization techniques in VMs and bare-metal systems [6], [36], [37]. The scheduling of applications in containers is managed by the host OS, which means achieving full isolation from interrupts is not possible. Consequently, optimizations such as a tickless-kernel, and isolation of selected CPUs is not feasible as a shared kernel is used requiring access to the specific cores. Due to this difficulties in isolating containers, it is essential to explore new approaches and conduct assessments of the suitability of these optimizations in containers.

To minimize the impact of the host OS on the container and between containers, LXC provides a method for reserving cores and memory exclusively. Additionally, automatic load balancing in LXC can be disabled to reduce overhead and make sure no additional scheduling is needed [3]. To summarize, these specialized container isolation techniques help to minimize the external influence on a container.

Fine-tuning the poll mode for idle CPUs, disabling energy-saving mechanisms, and turning off audit messages, can improve container performance similar as with VMs. Interrupts affinity can be set to a specific core, and logging of backtraces can be reduced to improve latency. Disabling hyper-threading improves latency for all systems. Table I presents the suggested list of boot parameters. The list is based on the presented container adoptions of optimizations for VMs in the study of Gallenmüller et al. [21]. Using the program *taskset*, it is possible to pin the affinity of all ready-copy-update processes to a core to reduce their scheduling on container cores. However, when resources are limited, the CPUs should be shared between containers leading to increased tail latencies.

V. Measurement Setup

To automate management and synchronization of testbed resources while performing experiments on bare-metal, VMs, and containers, we use the Plain Orchestrating Service by Gallenmüller et al. [38]. This enables the execution of synchronized experiments on hardware and VMs. Using IPMI for controlling the state of the machines enables flexibility and simplifies the execution of the same experiments using packet-processing on bare-metal and VMs. Utilizing identical scripts improves comparability among containers, VMs, and bare-metal environments.

The used software VirtualBMC [20] to control VMs using IPMI uses libvirt-attached VMs and enables the control of the boot process, setting of live-boot images, and boot parameters, which permits the reuse of scripts originally designed for bare-metal on VMs. However, VirtualBMC do not provide an interface for containers. Other software to control containers such as Docker Engine or Kubernetes cannot be used as they can only control containers but not hardware machines and VMs. To incorporate LXC into our ecosystem, we developed a modified version of VirtualBMC called *VirtualLXCBMC* including support for LXC functionality such as booting, shutdown, or restart. Since containers use a different image format, live images must be configured using LXC directly. However, the steps taken after startup are similar to those in VMs. To facilitate replication of our results, the *VirtualLXCBMC* is made available as open-source software alongside this paper.

For precise measurements, load generation and timestamping are performed externally, as depicted in Figure 2. The load-generator (LoadGen) features an Intel Xeon Silver 4116 CPU, 192 GB RAM, and a dual-port Intel 82599ES 10-Gigabit SFP+ NIC connected to the Device-under-Test (DuT) using optical fibers. The DuT is equipped with an AMD EPYC 7551P 32-Core Processor, 128 GB RAM, and 2 × Intel X710 10GbE SFP+ NICs, where one port each is linked to the LoadGen. On the DuT, we execute our experiments using bare-metal, VM, or container. The interfaces are directly accessed from within the used solution. To ensure high-precision measurements per packet at line-rate we used a timestamping machine (timestamper) linked to the fibers between DuT and LoadGen with passive optical terminal access points (TAPs). These TAPs introduce a constant delay to the timestamps on both ends

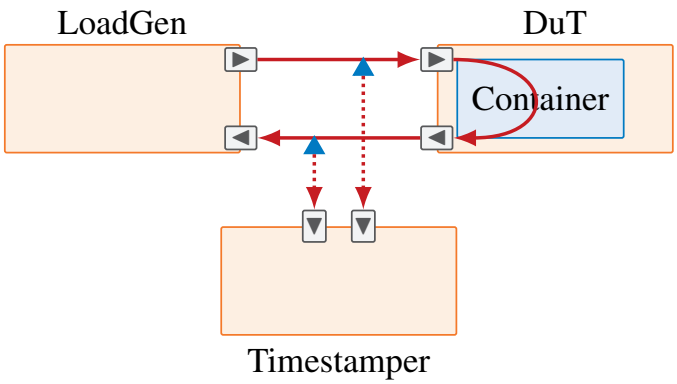

Figure 2. Measurement-setup structure

and can be neglected. The timestamper was equipped with an AMD EPYC 7542 32-Core Processor, 500 GB of memory, and an Intel E810-XXVDA4 25 Gbit/s NIC flashed to 10 Gbit/s offering a precision of 1.25 ns [39]. This is used to take timestamps in the hardware using the precision of the Intel E810 NIC. Timestamps taken in software are prone to the same interrupts and kernel operations as our DuT therefore using hardware timestamping reduces significantly the influence of the timestamping method.

This configuration facilitates precise analysis of packet processing tail-latency. We used PostgreSQL for evaluation to enable easy extension, and evaluating of the analysis. Further, we employed MoonSniff scripts [35] of MoonGen on LoadGen and timestamper to transmit and record minimally sized packets with 64 B. We assigned identifier numbers to transmitted packets for the purpose of correlation, and the recorded packets were timestamped using the respective hardware timestamping features. Previous studies suggests that the primary factor relevant for packet processing is the number of packets, and not their size [16], [18].

We use Debian Bullseye 11 (kernel 5.10) and execute a libmoon [35] layer two forwarding application to minimize the impact of the application itself, unless specified otherwise. The forwarding application runs inside the container. All experiments employ the packet rates from 10-1000 kpackets/s to analyze the effect on latencies. The number of data points collected depends on the packet rate, for example with 1 Mpkts/s we collect 160 million data points per experiment in 160 s.

VI. Evaluation

Our evaluation of tail latency behavior in packet-processing containers, focuses on optimizations described in Section IV including OS kernel variants, such as a real-time (RT) kernel and a vanilla kernel. We assess packet rate behavior and compare the outcomes to those of VMs and bare-metal.

A. Scenario

To examine low-latency behavior in container setups, we devised a straightforward scenario depicted in Figure 2. Packets are generated externally, transmitted to the DuT, and forwarded

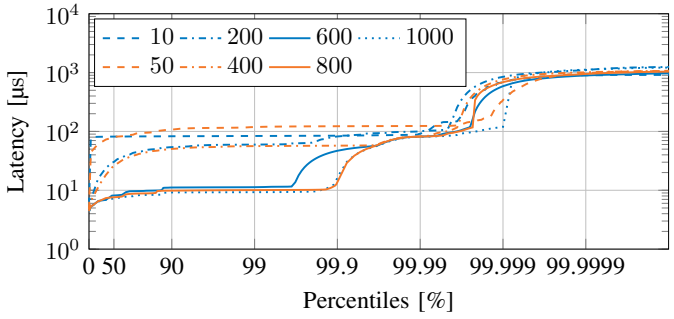

Figure 3. HDR diagram of latency for selected packet rates (kpkts/s) on the vanilla Linux kernel without optimizations

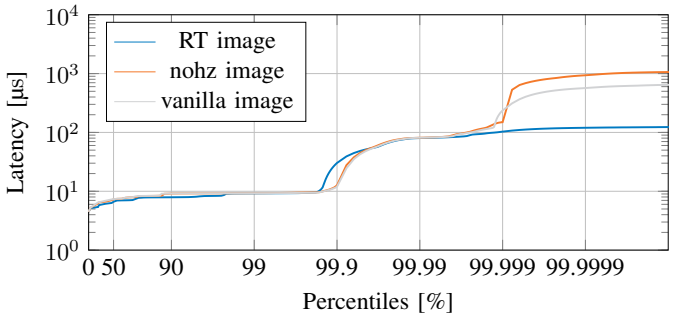

Figure 5. HDR diagram of non-optimized variants of Debian kernel at 1 Mpkt/s

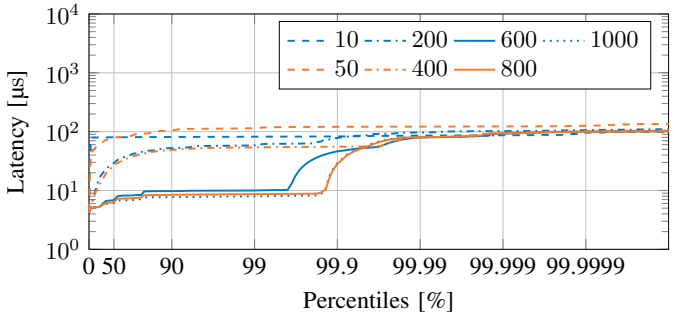

Figure 4. HDR diagram of latency for selected packet rates (kpkts/s) on the optimized Linux RT kernel

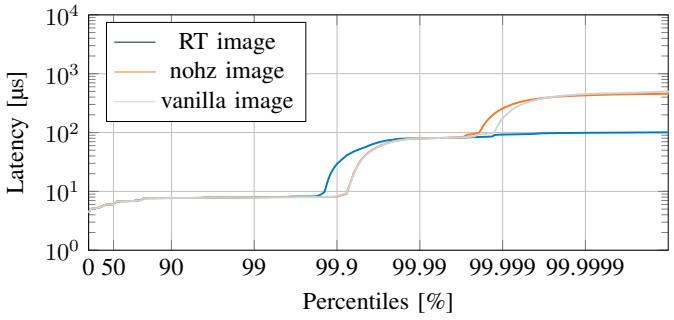

Figure 6. HDR diagram of optimized variants of Debian kernel at 1 Mpkt/s

through a basic packet processing application before being sent back over another link. The application operates within the analyzed system.

We employ a libmoon l2 forwarding application to examine the latency induced by network processing and virtualization rather than by the application itself. This approach enables the evaluation of the effect of optimization techniques in isolation. We examine the same application on VMs and bare-metal to compare our findings. Through these results, we provide recommendations for using low-latency applications in containers and identify any limitations.

To establish the validity of our findings, we repeated all experiments multiple times, selecting the measurement with the worst tail-latency for evaluations to ensure the capture of rare events.

B. Packet Rates

We analyze the latency by comparing the effects on systems with and without optimizations discussed in Section IV starting with comparing different packet rates. Figure 3 illustrates the behavior of a non-optimized vanilla compared to the optimized RT variant presented in Figure 4. The logarithmic plots show the latency against the percentiles using high-dynamic-range (HDR) diagrams [40]. Using HDRs allows to focus on tail latency events to analyze even rare latency spikes. We focus on tail latencies and rare events during our evaluation using HDR diagram plots (e.g. Figures 3 and 4). Across all the rates, the optimized and non-optimized systems exhibit similar tail-latency behavior at the 99.99^{th} percentile and 99.9995^{th} percentile, respectively. Lower packet rates are more susceptible to high latencies in lower percentiles; for instance, at 200 kpkts/s, more than 50 % of all packets reach a maximum latency of 110 µs. Additionally, all measurements indicate higher median latencies when the packet rate is decreased which is suspect to the lower number of measured packets.

At lower packet rates, the impact on percentiles for rare events is higher because fewer packets are captured within the same measurement time. Hence, we conclude that rare occurrences have a higher impact at lower-rates, no further observations of reasons for the system were made. Therefore, higher packet rates are more suitable for tail-latency analysis, whereas higher rates only are insufficient for median analysis due to the significant differences in median behavior. Overall, measurements indicate that all evaluated variants can process 1 Mpkts/s with minimally sized packets used further for detailed evaluations.

C. Optimizations

Additionally, we investigate the difference in OS kernel variants by including experiments with vanilla, *nohz*, and RT kernel. The *nohz* kernel is using a kernel patch allowing to remove timer ticks from cores with only one application, whereas the RT kernel provides deterministic behavior which should improve latencies constant behavior.

The results of the three kernel variants are presented in Figures 5 and 6. Figure 5 shows the measurement results the

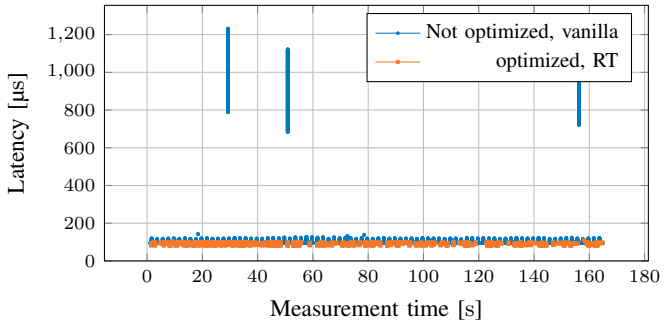

Figure 7. 5000 worst latency events for measurements using LXC-containers based on Debian 11

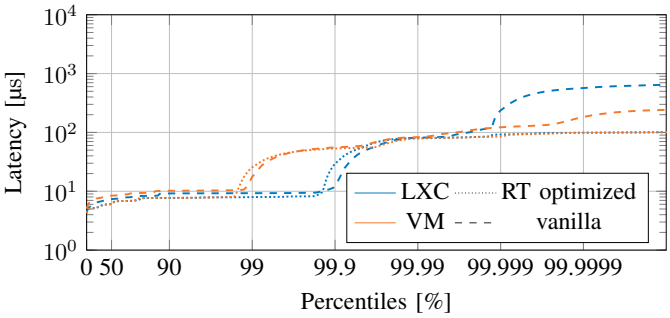

Figure 8. HDR diagram of latency on VMs and LXC containers

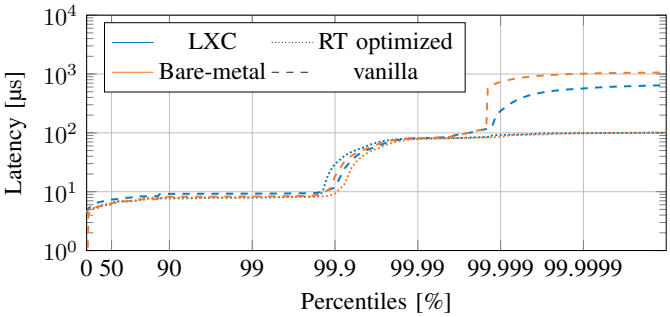

Figure 9. HDR diagram of latency from measurements using bare-metal and LXC containers

non-optimized OS in the three kernel variants, revealing a high tail-latency spike toward 1000 µs at the 99.999^{th} percentile when using the *nohz* and vanilla kernel. All kernel variants exhibit latency spikes at the 99.9^{th} percentile. The RT kernel variant records a maximum latency of around 140 µs. In contrast, Figure 6 shows the results for the optimized OS, where the RT variant had a slightly lower maximum latency of 110 µs compared to the non-optimized RT variant. Moreover, the *nohz* and vanilla kernel variants led to significantly lower tail latencies at around 510 µs compared to the measurements using kernels without optimization.

Concerning tail latency, the optimized outperform the non-optimized variants affected by interrupts and rescheduling. Similar to the findings of related work on bare-metal [21], our measurements indicate that the *nohz* variant attains nearly the same tail latencies as the vanilla one on Debian 11. Using the RT variant displays similar results compared to the vanilla one up to 99.99^{th} percentile of latencies. However, latencies do not increase further, which limits the tail latency. The *nohz* kernel can only provide benefits when the no scheduling is needed, but LXC requires this to schedule the container engine on the relevant cores. Whereas the RT kernel provides more stable tail-latencies due to the deterministic behavior of the OS kernel operations.

In Figure 7, the 5000 worst-case events of our baseline measurements using the non-optimized vanilla variant are compared to those of an optimized RT one. The behavior of the optimized RT kernel variant is similar to that of the non-optimized vanilla kernel variant without the rare, significant outliers which are caused by rare interrupts scheduled on the packet processing core.

D. Container vs. VMs

Figure 8 presents a comparison of measurements between container and VM setups using an optimized RT and a non-optimized vanilla variant. The forwarding application for this experiment is executed inside the VM compared to inside the container. The VMs are operating as KVM instances on the DuT. In our measurements, the RT kernel variant demonstrates better results, which we attribute to the utilization of Debian 11 in contrast to Debian 10 used in related work [2], [6].

As presented in Figure 8, the performance differences between VMs and containers are insignificant concerning tail latencies. With VMs in the vanilla variant showing lower tail-latencies due to a better isolation by default. However, VMs with higher overhead reach the same maximum tail latency at lower percentiles. By carefully isolating and optimizing containers, similar results compared to VMs can be achieved, requiring fewer resources.

E. Container vs. Bare-metal

For this comparison, the forwarding application is executed directly on bare-metal compared to running inside a container. The vanilla variant for containers, which provides minimal container isolation by default, yields slightly lower tail latencies (Figure 9). Meanwhile, optimized bare-metal experiments slightly outperform optimized containers primarily due to a higher degree of isolating the forwarding application from interrupts. Our findings differ from those reported in [21] as we use a different mainboard. Specifically, [21] used an Intel mainboard with Intel CAT for pinning the cache to a specific core, which is currently not supported by AMD. Furthermore, they used a DPDK l2fwd application without Lua as wrapper, which can result in additional latencies caused by the wrapper. Although bare-metal configurations are generally preferable due to better resource isolation, containers may be used when resource sharing is necessary.

To conclude, containers exhibit only a minimal overhead compared to bare-metal, contingent on the underlying hard-

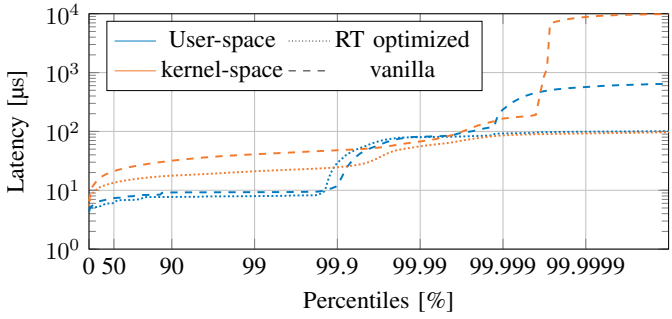

Figure 10. HDR diagram of latency showing measurements on kernel- versus user-space packet processing

ware, which may vary. Consequently, hardware selection is critical when ultra low-latencies are required.

F. User-space vs. Kernel-space Network Driver on Container

Thus far, networking in user-space have been examined. For comparison with kernel-space networking, we investigated a Linux traffic control mirroring application. The application enables the assessment of the impact of kernel-space networking.

Figure 10 shows the tail latency for kernel-space networking compared to user-space networking.The tail latency after the 99.999^{th} percentile was similar for both variants of packet processing. The optimized variants measured a maximum latency of 110 µs in experiments. Compared to the 99.8^{th} percentile for user-space packet processing the latency increases gradually for the 50^{th} percentile. This suggests that the optimizations employed for user-space can be used for kernel-space processing. Thus, container isolation enables the use of kernel drivers for low-latency applications.

In the non-optimized variant, the tail latency in kernel-space networking exhibits a worse performance, with a difference of one order of magnitude. However, comparing kernel- and user-space packet processing in containers reveals that both are valuable for low-latency applications in containers, provided that optimization techniques are carefully used. Evaluating in general using user-space networking was clearly an advantage as it outperforms kernel-space-networking in the non-optimized variant and the tail-latency in the optimized variant do not differ significantly.

VII. Tail Latency Model

We use EVT to model the tail latency behavior of each measurement. Each model is generated using the first 20 % of measurement datapoints and validated using the following 80 %. The model is derived using the peaks-over-threshold method with a threshold set at the 99.95^{th} percentile value of measured latencies. Then, it is fitted to a Generalized Pareto Distribution (GPD) using a maximum likelihood estimator with a confidence level of 95 %. This model can be used to calculate the return level x_m for an arbitrary return period m, as shown in Equation (1), with the threshold μ, scale σ, tail ξ, and the proportion of latencies larger than the threshold $\frac{D_{d>\mu}}{D}$. The return level represents the magnitude of latencies, which is expected to be exceeded precisely once during the return period [30]. It is effectively the expected worst-case latency during a given period. This can be seen as a complementary measure to worst-case latency bounds obtainable by analytical methods, such as network calculus. The limit of this function for $m \to \infty$ describes the behavior of the tail latencies on an arbitrarily long time horizon, signifying convergence or divergence. We validated the models by comparing the return levels and their convergence behaviors to the remaining 80 % of the data, predicting tail latencies on a four-fold time horizon.

$$x_m = \mu + \frac{\sigma}{\xi} \cdot \left[\left(m \cdot \frac{D_{d>\mu}}{D} \right)^{\xi} - 1 \right] \tag{1}$$

Table II shows the percentage of converging models ($x_m \to$) and the number of return-level exceedances ($\mu \uparrow$) for each platform and variant. A high proportion of converging models indicates that the data is more appropriate to be represented by this type of statistical model. We can observe that containers using the RT variant have the highest percentage of converging models at 66.7%. While this is not an optimal result, it shows significant improvement over the other variants. Further improvements might be gained by tweaking the threshold value. The mean return-level exceedances should ideally be equal to 1, indicating a model with good predictive capabilities. The analysis indicates that containers using a vanilla variant exhibit the best mean exceedance behavior.

Table II
Comparison of EVT predictions for each platform and image. Shows the percentage of converging models ($x_m \to$) and mean number of return level exceedances ($\mu \uparrow$). Convergence is better the closer to 100 %, exceedances are better the closer to 1.

Platform	Image			
	Vanilla		opt. RT	
	$x_m \to$	$\mu \uparrow$	$x_m \to$	$\mu \uparrow$
Bare-metal	16.7 %	0.33	60.0 %	3.30
Virtual Machine	25.0 %	2.58	58.3 %	4.00
Container	16.7 %	1.50	**66.7** %	3.83

Table III
Tail-latency values for non-optimized vanilla and optimized RT version for all three systems with user-space networking and kernel-space networking on containers only

Technology	non-optimized vanilla	optimized RT
container		
kernel-space	9969.08 µs	100.19 µs
user-space	659.25 µs	108.86 µs
VM	840.63 µs	124.12 µs
Bare-metal	1077.44 µs	101.81 µs

Table IV
SUMMARIZED RECOMMENDATIONS WITH RANKS FOR EACH CATEGORY FROM ✓✓✓ (BEST) TO ✗✗✗ (WORST).

Technology	Latency	Security	Resources
VM			
optimized	✓	✓✓	✗
non-optimized	✗✗✗	✓✓	✓
Container			
optimized	✓✓	✓	✓✓
non-optimized	✗	✓	✓✓✓
Bare-metal			
optimized	✓✓✓	✓✓✓	✗✗
non-optimized	✗✗	✓✓✓	✗✗

VIII. RECOMMENDATIONS FOR LOW-LATENCY-SLICED APPLICATIONS

Table III presents the tail-latency of the optimized RT and the non-optimized vanilla variant for each technology. We recommend a top-down strategy for the choice of a system for URLLC based on the presented findings. While a bare-metal solution is best suited for commodity hardware, it can be resource-intensive and challenging to respond to on-the-fly demands. Our measurements herein demonstrated that VMs and containers perform similarly, therefore, we recommend to analyze additional aspects such as security and resource usage. We recommend the use of containers to ensure reduced resource usage and high flexibility. However, if higher security and isolation of applications are required, VMs are recommended. Containers and VMs can be hosted on the same hardware system providing both to customers.

Table IV summarizes the recommendations. We have ranked the categories for non-optimized and optimized VMs, containers, and bare-metal solutions. Generally, optimized variants perform better than non-optimized variants in all systems, whereas the choice of technology ultimately depends on the infrastructure, requirements, and available resources.

IX. LIMITATIONS

Our analysis of virtualization overhead was based on single instances and not concurrent ones. Only simple applications were analyzed since our focus was technology induced latency.

We did not examine shared network resources, such as previous studies [6] and [2] have analyzed for VMs. Further studies are necessary to explore potential improvements and provide a more in-depth analysis of shared system resources.

Our analysis focuses exclusively on the behavior of LXC containers, since previous studies indicate that alternative solutions introduce additional overhead [8], [12]. However, this has not been verified further, and comparing container solutions for improving latency is part of future research.

X. REPRODUCIBILITY

The scripts, raw data, and analyses results required to reproduce our findings are available online, including the raw PCAPs, the data extracted from these PCAPs, and the plots obtained for each measurement[1]. The scripts enable the reproduction of all measurements and calculations on other systems, provided that the necessary hardware is available for per-packet timestamping along with passive optical TAPs.

XI. CONCLUSION AND FUTURE WORK

The need for reliable and predictable low-latency is critical in applications such as autonomous driving or remote medical procedures. Additionally, resource sharing and on-demand service provisioning, such as network slices in 5G, are also vital. This study demonstrates that lightweight virtualization is a suitable alternative for high-reliability, low-latency applications. However, achieving this requires tactful selection of optimization parameters, such as using rt-kernels.

We can use a user-space networking application to achieve high reliability and significantly reduce tail latency. Furthermore, VMs and containers exhibit comparable performance; however, containers require fewer resources but cannot be completely isolated. We employed an EVT model to assess the predictability of tail latencies in containers. It was found that an optimized system in container with an RT kernel is more converging in models than any configuration based on VMs or bare-metal. This study presents the first in-depth analysis of packet-processing latency in containers.

Further, we plan to analyze the influence of concurrent containers, CPU resource sharing, and evaluate the potential of SR-IOV-based NIC sharing to improve low-latency and high availability of resources. We also plan to enhance predictability using statistical methods over time to enable accurate planning. Finally, we plan to investigate and compare different container solutions and their latencies beyond current findings as well as the effect on using VMs and containers on the same system.

ACKNOWLEDGMENTS

This work was supported in part by the European Union Horizon 2020 research and innovation programme (project SLICES-SC, 101008468, and SLICES-PP, 101079774), the Bavarian Ministry of Economic Affairs, Regional Development and Energy (project 6G Future Lab Bavaria), and the German Federal Ministry of Education and Research (project 6G-life, 16KISK002, and project 6G-ANNA, 16KISK107).

REFERENCES

[1] "5G: Study on Scenarios and Requirements for Next Generation Access Technologies," 2017, Last accessed: May 22, 2023. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/138900_138999/138913/17.00.00_60/tr_138913v170000p.pdf

[2] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, "5G QoS: Impact of Security Functions on Latency," in *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 2020, pp. 1–9.

[3] RedHat, "cpuset - Red Hat Enterprise Linux 6," 2023, Last accessed: May 22, 2023. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/sec-cpuset

[1] https://wiednerf.github.io/containerized-low-latency/

[4] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance Overhead Comparison between Hypervisor and Container Based Virtualization," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, 2017, pp. 955–962.

[5] A. K. Yadav, M. L. Garg, and Ritika, "Docker containers versus virtual machine-based virtualization," in *Emerging Technologies in Data Mining and Information Security*, A. Abraham, P. Dutta, J. K. Mandal, A. Bhattacharya, and S. Dutta, Eds. Singapore: Springer Singapore, 2019, pp. 141–150.

[6] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "Ducked Tails: Trimming the Tail Latency of(f) Packet Processing Systems," in *17th International Conference on Network and Service Management, CNSM 2021, Izmir, Turkey, October 25-29, 2021*, P. Chemouil, M. Ulema, S. Clayman, M. Sayit, C. Çetinkaya, and S. Secci, Eds. IEEE, 2021, pp. 537–543.

[7] D. Gedia and L. Perigo, "Performance evaluation of sdn-vnf in virtual machine and container," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2018, pp. 1–7.

[8] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, M. L. Scott and L. L. Peterson, Eds. ACM, 2003, pp. 164–177.

[10] L. Abeni, C. Király, N. Li, and A. Bianco, "Tuning KVM¸ to enhance virtual routing performance," in *Proceedings of IEEE International Conference on Communications, ICC 2013, Budapest, Hungary, June 9-13, 2013*. IEEE, 2013, pp. 3803–3808.

[11] M.-N. Tran and Y. Kim, "Network Performance Benchmarking for Containerized Infrastructure in NFV environment," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, Jun. 2022, pp. 115–120.

[12] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393.

[13] J.-G. Cha and S. W. Kim, "Design and Evaluation of Container-based Networking for Low-latency Edge Services," in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct. 2021, pp. 1287–1289.

[14] S. Lin, P. Cao, T. Huang, S. Zhao, Q. Tian, Q. Wu, D. Han, X. Wang, and C. Zhou, "Xmasq: Low-overhead container overlay network based on ebpf," *CoRR*, vol. abs/2305.05455, 2023.

[15] D. Zhuo, K. Zhang, Y. Zhu, H. H. Liu, M. Rockett, A. Krishnamurthy, and T. Anderson, "Slim: OS kernel support for a Low-Overhead container overlay network," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 331–344. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/zhuo

[16] T. Zhang, L. Linguaglossa, J. Roberts, L. Iannone, M. Gallo, and P. Giaccone, "A benchmarking methodology for evaluating software switch performance for NFV," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, Jun. 2019, pp. 251–253.

[17] G. K. Lockwood, M. Tatineni, and R. Wagner, "SR-IOV: Performance Benefits for Virtualized Interconnects," in *Annual Conference of the Extreme Science and Engineering Discovery Environment, XSEDE '14, Atlanta, GA, USA - July 13 - 18, 2014*, S. A. Lathrop and J. Alameda, Eds. ACM, 2014, pp. 47:1–47:7.

[18] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, Apr. 2010, pp. 1–12.

[19] Intel Corporation, Oct 2013. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html

[20] "How to use virtualbmc," Jul 2020. [Online]. Available: https://docs.openstack.org/virtualbmc/latest/user/index.html

[21] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "How Low Can You Go? A Limbo Dance for Low-Latency Network Functions," *Journal of Network and Systems Management*, vol. 31, no. 20, Dec. 2022.

[22] D. Bozilov, M. Knezevic, and V. Nikov, "Optimized threshold implementations: securing cryptographic accelerators for low-energy and low-latency applications," *J. Cryptogr. Eng.*, vol. 12, no. 1, pp. 15–51, 2022.

[23] V. Jain, H.-T. Chu, S. Qi, C.-A. Lee, H.-C. Chang, C.-Y. Hsieh, K. K. Ramakrishnan, and J.-C. Chen, "L25GC: A Low Latency 5G Core Network Based on High-Performance NFV Platforms," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 143–157.

[24] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav, "Quartz: A New Design Element for Low-Latency DCNs," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, p. 283–294, aug 2014.

[25] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Comput. Surv.*, vol. 53, no. 2, mar 2020.

[26] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud Scale Load Balancing with Hardware and Software," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, p. 27–38, aug 2014.

[27] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 29–42.

[28] DPDK Project, "Home - DPDK," 2023, Last accessed: May 22, 2023. [Online]. Available: https://www.dpdk.org/

[29] C.-N. Mao, M.-H. Huang, S. Padhy, S.-T. Wang, W.-C. Chung, Y.-C. Chung, and C.-H. Hsu, "Minimizing Latency of Real-Time Container Cloud for Software Radio Access Networks," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2015, pp. 611–616.

[30] S. Coles, J. Bawa, L. Trenner, and P. Dorazio, *An Introduction to Statistical Modeling of Extreme Values*. Springer, 2001, vol. 208.

[31] T. Hsing, "On Tail Index Estimation Using Dependent Data," *The Annals of Statistics*, vol. 19, no. 3, pp. 1547–1569, 1991.

[32] M. Helm, F. Wiedner, and G. Carle, "Flow-level Tail Latency Estimation and Verification based on Extreme Value Theory," in *18th International Conference on Network and Service Management, CNSM 2022, Thessaloniki, Greece, October 31 - Nov. 4, 2022*, M. Charalambides, P. Papadimitriou, W. Cerroni, S. S. Kanhere, and L. Mamatas, Eds. IEEE, 2022, pp. 359–363.

[33] N. Mehrnia and S. Coleri, "Wireless Channel Modeling Based on Extreme Value Theory for Ultra-Reliable Communications," *IEEE Trans. Wirel. Commun.*, vol. 21, no. 2, pp. 1064–1076, 2022.

[34] M. Bennis, M. Debbah, and H. V. Poor, "Ultrareliable and Low-Latency Wireless Communication: Tail, Risk, and Scale," *Proc. IEEE*, vol. 106, no. 10, pp. 1834–1853, 2018.

[35] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, K. Cho, K. Fukuda, V. S. Pai, and N. Spring, Eds. ACM, 2015, pp. 275–287.

[36] AMD, "Performance Tuning Guidelines for Low Latency Response on AMD EPYC-Based Servers Application Note," Jun. 2018, Last accessed: May 22, 2023. [Online]. Available: http://developer.amd.com/wp-content/resources/56263-Performance-Tuning-Guidelines-PUB.pdf

[37] J. Mario and J. Eder, "Low Latency Performance Tuning for Red Hat Enterprise Linux 7," Nov. 2017, Last accessed: May 22, 2023. [Online]. Available: https://access.redhat.com/sites/default/files/attachments/201501-perf-brief-low-latency-tuning-rhel7-v2.1.pdf

[38] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos framework: a methodology and toolchain for reproducible network experiments," in *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Virtual Event, Munich, Germany, December 7 - 10, 2021*, G. Carle and J. Ott, Eds. ACM, 2021, pp. 259–266.

[39] Intel Corporation, "E810 datasheet rev2.5." [Online]. Available: https://cdrdv2-public.intel.com/613875/613875_E810_Datasheet_Rev2.5.pdf

[40] G. Tene, "HDRHistogram: A High Dynamic Range Histogram," Aug. 2021, Last accessed: May 22, 2023. [Online]. Available: http://hdrhistogram.org/