Applicability of Hardware-Supported Containers in Low-Latency Networking

Alexander Daichendt, Florian Wiedner, Jonas Andre, Georg Carle
Technical University of Munich
Garching, Germany
{daichend, wiedner, andre, carle}@net.in.tum.de

***Abstract*—Containers share the kernel with the host OS, which has implications for the network stack. Achieving connectivity between containers exclusively in software is unsuitable for reliable, low-latency applications. While extensive research has been conducted on virtual machines processing real-time traffic with hardware support, the impact of network latencies in containerized environments has received comparatively less attention. This paper analyzes throughput and network latencies in container topologies on a single host featuring single-root input/output virtualization, Linux Containers, and commercial off-the-shelf hardware. Using a state-of-the-art timestamping methodology, we measure latencies with a resolution of $1.25\,\mu s$ without introducing delay by the measurement methodology itself. We evaluate a single flow in a line topology with up to 64 containers. The experiments demonstrate that pinning interrupt request handlers to non-uniform memory access nodes increases throughput and decreases latencies. Furthermore, we identify dTLB misses, rescheduling interrupts, and soft interrupt floods as critical challenges as they cause spikes in latencies, and isolation is impossible. This paper contributes findings to minimize bottlenecks and limitations for real-time container applications.**

***Index Terms*—low latency, container, lxc, virtualization, numa, single-root input/output virtualization, nfv**

I. Introduction

Virtualization is a powerful tool that enables resource sharing and on-demand provisioning. Testing and verifying in isolated environments mitigates potential issues at later stages of the development process. For instance, Internet of Things manufacturers may seek to ensure their new mesh protocol can scale while maintaining latency assurances. Similarly, a researcher in high-performance computing validates their network stack before utilizing expensive resources.

Manually wiring hardware for deployments may be feasible for small systems without on-demand requirements. When the scale of the system increases, the complexity of setup and maintenance grows, and human error becomes more likely. Network virtualization and resource sharing are ways to minimize costs and errors by enabling the automation of setup and provisioning procedures.

Virtual machines (VMs) provide comprehensive isolation by virtualizing a complete operating system (OS), including the kernel. The hypervisor, a software layer, offers controlled access and abstracted interfaces for VMs to access physical resources. Due to this nature, VMs are heavy-weight and require more resources than non-virtualized systems. In contrast, containers leverage kernel-level features to establish an isolated sandbox at process level and share the kernel with the host OS as lightweight alternative.

Wiedner et al. [1] demonstrated that virtual, software-based networking, as used in the network emulator Mininet, cannot achieve stable, low latencies due to the unpredictable nature of virtual networking. By utilizing containers with physical networking equipment, we can leverage the benefits of low overhead virtualization and low latencies. This paper analyzes throughput and latencies by creating networking topologies on a single host featuring single-root input/output virtualization (SR-IOV), Linux containers (LXC), and commercial off-the-shelf (COTS) hardware. We evaluate a single-flow-line topology to identify bottlenecks.

The main contributions of this paper are:

1) pinning interrupt request handlers (IRQs) to non-uniform memory access (NUMA) nodes increases throughput and decreases latencies,
2) insight into bottlenecks and limitations: dTLB misses, rescheduling interrupts, and soft interrupt floods,
3) recommendation for the usage of containers and VMs including recommondations towards the technical use of CPU core pinning.

The remaining paper is structured as follows: Sections II and III introduces the state-of-the-art and methodology. Section IV evaluate and discuss the acquired data. In Section V, we recommend scenarios in which containers are an efficient replacement for VMs and provide reproducibility information in Section VI. Section VII outlines limitations and Section VIII concludes the paper.

II. Background and Related Work

This section provides a comprehensive overview of background information and related work focusing on containers, SR-IOV, and the Linux network stack.

A. Virtualization Techniques

This work utilizes LXC due to its minimalistic and lightweight approach compared to other solutions. LXC performs marginally better than Docker, as shown by Morabito et al. [2], but lacks convenience features.

©IFIP, 2024. This is the author's version of the work.
It is posted here by permission of IFIP for personal use. Not for redistribution.
The definitive version was published in CNSM conference proceedings.

Table I
SUMMARY OF I/O PERFORMANCE CHARACTERISTICS OF VIRTUALIZATION IN THE STATE-OF-THE-ART

Technique	Sources	Latency	Throughput
KVM VM	[3]–[6]	✓	✓✓✓
Docker	[2], [3]	✓✓	✓✓
LXC	[2], [4]–[6]	✓✓✓	✓✓✓

Felter et al. [3] showed that a Docker container, a containerized environment that provides separation between different applications, outperforms KVM-based VMs in MySQL throughput. According to them, compute and memory access are nearly overhead-free, while OS interactions and I/O cause overhead. Sharma et al. [4] came to similar conclusions. They found LXC to be 2% slower than bare metal across multiple benchmarks. Moreover, they measured VMs to be 3% slower than LXC, although the performance in I/O-bound processes was inferior. For example, disk latency was eight times higher in VMs than in LXC. Bessera et al. [5] confirmed the assessment. Furthermore, Sharma et al. [4] revealed that LXC, compared to VMs, is more susceptible to noisy neighbors or adversarial workloads, leading to resource deprivation for co-located containers. These findings indicate that in I/O-bound and memory-intensive workloads, containers outperform VMs. The performance characteristics are summarized in Table I.

B. Single-Root Input/Output Virtualization

SR-IOV is an extension to the PCI specification, enabling physical devices to present multiple lightweight virtual functions (VFs) to the PCIe bus and the OS. The device's physical function (PF) serves as a fully-featured PCI function for management and configuration. A VF has separate send and receive queues and shares the underlying physical resources. It can interrupt independently from other VFs and the PF, resulting in minimal overhead, near line-rate throughput, and low network latency, as shown by numerous studies [7]–[9].

Dong et al. [10] demonstrated that SR-IOV significantly reduces CPU utilization and achieves line-rate throughput scaling nearly perfectly with up to 60 VMs. Liu [8] benchmarked SR-IOV against bare metal and virtio—a virtual network device. Compared to bare metal, SR-IOV exhibited slightly higher network latencies caused by interrupt virtualization. Virtio performed roughly 5 times worse than SR-IOV, indicating that this technique is not suitable for real-time applications.

Before SR-IOV was introduced on COTS hardware, achieving low-latency networking required passing a physical device through to a VM. However, this approach is not scalable due to limited PCIe lanes. As expected, passthrough offers lower network latencies than SR-IOV by bypassing the logic required to pass a frame to the correct VF [7].

C. Network Stack of the Linux Kernel

When a network interface card (NIC) receives a packet, it is transferred via direct memory access (DMA) from the PCIe bus into the main memory. A DMA descriptor determines the main memory location in the receive (RX) queue, a ring buffer holding pointers to physical memory [11]. Packet loss may occur if processing is not quick enough, i.e., during packet bursts, causing the hardware to overwrite data in the ring buffer.

Whenever data is copied into the main memory, the NIC raises a hardware IRQ [12]. The CPU core handling the IRQ can be configured with affinities; otherwise, the OS scheduler decides. The NIC cannot raise another hardware IRQ until the current one is cleared. Hence, work in the hardware IRQ handler is kept to a minimum. The driver schedules a software IRQ (softirq) to process the packets further and clear the hardware IRQ, allowing the NIC to raise another one. Data are copied and processed layer by layer further until they can be consumed by the application or forwarded.

Modern NICs can have multiple receive (RX) and transmit (TX) queues. Incoming packets are distributed using Receive Side Scaling (RSS). RSS calculates a hash based on header information to determine the processing queue [13]. Therefore, packets of a flow are handled in the same queue to avoid out-of-order processing. Each queue has its own hardware IRQ, and the driver can the merge them.

With an increasing amount of arriving packets, more time has to be spent handling interrupts. As a mitigation, the Linux kernel implements busy polling, a form of interrupt coalescence, with the New API (NAPI). After reaching a certain threshold of packets per second, the driver switches into poll mode and turns off hardware IRQs [14]. A separate thread periodically polls the RX queue for new packets. NAPI significantly lowers the network latencies and increases throughput on busy systems, as shown by Salim et al. [15].

Beifuß et al. [16] created a model to predict latencies introduced by the Linux kernel and drivers and compared it against measurements. They observed that the interrupt rate decreased as the packet rate increased, indicating the successful transition to NAPI.

D. Network Performance with Containers

Xavier et al. [17] showed that networking within LXC with network namespaces achieves the highest throughput and lowest latencies out of selected network isolation mechanisms, including Xen and OpenVZ. While Linux-VServer performed identically to bare metal, it provided less isolation due to only sorting frames based on an identifier.

Ara et al. [18] evaluated the performance of software switches by using LXC on the sending and receiving side. They compared kernel networking with VPP and SR-IOV, concluding that SR-IOV excels the others in throughput. Software solutions outperformed SR-IOV on a single host for latency, while SR-IOV exhibited lower latency in multi-host scenarios. However, their latency measurement was limited to mean values.

Rathore et al. [19] compared the performance of KVM and LXC containers as virtual routers. They benchmarked up to 8 concurrent VMs or containers regarding latency and throughput with SR-IOV and kernel networking. Although

LXC achieves higher packet rates and slightly lower latencies than KVM, they advise against using containers; kernel networking is less isolated, and limiting CPU utilization spent on the network stack per container is impossible.

Based on our findings in the state-of-the-art literature, the analysis of low-latency packet processing within containers just started with unanswered questions such as concurrent behavior, hardware influences, and real-world applications. This gap leads to further analysis of the effect of containers on traffic with low-latency requirements. In this paper, we especially analyze containers' concurrent behavior and latency's corresponding influence.

III. Measurement Methodology and Setup

The network performance of containers is evaluated with three hosts: timestamping, load generation (LoadGen), and the device under test (DuT) running containers. Figure 1 depicts the setup derived from [1]. Using separate hosts eliminates the influence of the measurement process itself on the results; latencies are measured precisely and accurately.

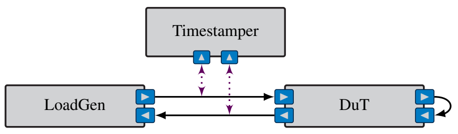

Figure 1. Experiment setup based on HVNet [1]

A. Overview

The LoadGen generates 64 B sized UDP datagrams using MoonGen [20], a high-performance packet generator. The machine has an Intel Xeon Silver 4116, 192 GB RAM with a dual-port Intel 82599ES 10-Gigabit SFP+ NIC connected to the DuT via optical fibers.

The DuT hosts LXC topologies on Debian 10 with real-time (RT) patches. According to Gallenmüller et al. [21], an RT kernel is preferable over others when cores are shared. The DuT features an AMD EPYC 7551P, 128 GB RAM, and two interconnected dual-port Intel X710 10Gbe SFP+ NICs. One port on each NIC is connected to the ingress and egress of the LoadGen. We use LXC 4.0.6, packaged by Debian 11. Datagrams arrive at the ingress, are processed by the container topology, and are returned to the LoadGen.

The Timestamper is connected to the ingress and egress with passive optical terminal access points, introducing twice the same constant delay that is not visible in the latency calculations. Each packet carries a unique identifier, allowing the correlation of packets to calculate the exact latency through the topology. The calculations are performed by the MoonSniff framework [20]. This methodology allows gathering latency without inducing latency through the measurement process. The Timestamper is equipped with an AMD EPYC 7542, 512 GB RAM, and an Intel E810-XXVDA4 25-Gigabit NIC flashed to $10\,\mathrm{Gbit/s}$ providing $1.25\,\mathrm{ns}$ precision [22].

B. Optimizations

Several optimizations are applied to the OS, kernel of the DuT, and the containers. Related work [23], [24] suggested turning off energy-saving features, read-copy updates, and activating poll mode when a core is idle. By turning off logging, which lowers CPU usage, the latency for some packets can be lowered. Moreover, configuring IRQ affinity for core 0 steers interrupts to core 0, which is not processing packets. Furthermore, simultaneous multithreading is disabled, as it does not aid in handling interrupts [13]. Finally, the performance governor is set for all cores to prevent the CPU from scaling down its frequency.

The containers in the experiments have no additional application running; the kernel and network namespaces accomplish all packet processing. Each container controls an array of libraries, encompassing journaling, a DHCP client, an SSH server, and a dbus daemon, all of which demand CPU time. Limiting all containers' user and system slices to CPU core 0 reduces work unrelated to processing IRQ handlers.

C. Arbitrary Flow Injection

Simulating networking scenarios includes user-defined flows on arbitrary ingress and egress points. This objective can be accomplished by following the instructions by Wiedner et al. [1]. Their methodology involves SR-IOV on the ingress and egress interface of the DuT (cf. Figure 2) and passing a tuple of VF to each VM. When the ingress NIC receives frames, the frames are associated with the MAC address to the VF. Consequently, the NIC sets the upper bound for the number of nodes and logical connections supported to the maximum amount of VFs the NIC can provide, in our case up to 64 nodes with one connection per node.

D. Network Topologies on a Single Host

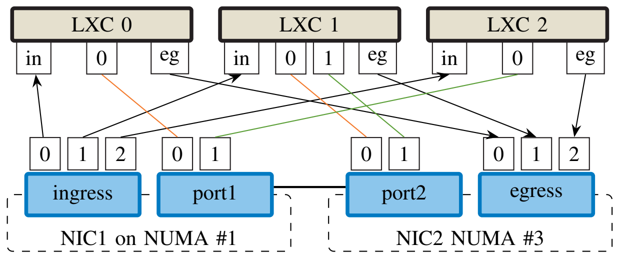

Figure 2. Overview of 3 hosts in line with SR-IOV on all NICs

To create virtual network topologies with real hardware, the remaining two ports of our Intel X710 NICs are connected via a DAC cable. For a logical connection between two nodes, we create one VF on each of the NICs and assign the same VLAN tag to them. We use policy routing, a feature of the Linux kernel [14], to route packets based on the UDP destination port. This setup ensures a packet is transmitted over the physical wire, introducing delay due to propagation and serialization on ingress and egress.

Figure 2 depicts how a logical line topology with length three is reflected by the physical hardware after combining the approach for flow injection in Section III-C and the setup with two interconnected NICs. Each container receives an ingress (in) and egress (eg) VF from the respective physical NIC. Two lines with the same color indicate a logical connection between two containers and are tagged with a VLAN ID. For instance, LXC0 and LXC1 are connected with the VF0 on `port1` and VF0 on `port2`. Any packet between the two containers is sent over the physical wire.

VFs from `port1` and `port2` are assigned to containers in a best-effort manner, resulting in assignments where memory must be copied between NUMA nodes. For example in Figure 2, a flow with the hops 0-1-2 has to cross NUMA nodes once to copy the packets into the egress on node 3.

IV. Evaluation

This section presents our evaluation of container in a line-topology. We begin by discussing the maximum achievable packet rate and highlight the bottlenecks. Subsequently, we evaluate the network latencies and discuss the results.

A. *Packet Rate*

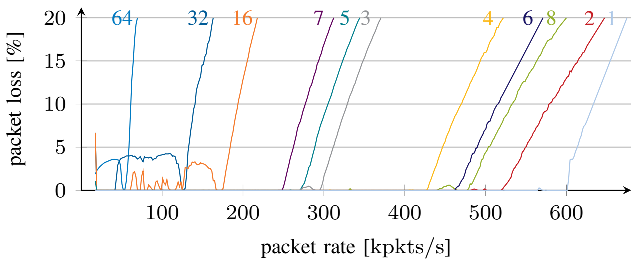

Figure 3. Packet loss at packet rate for line topologies with per node pinning

We restrict the number of RX and TX queues to one per interface, as additional queues yield minimal benefits for one flow [13]. To precisely measure the packet rate and loss, we increase the packet rate over time constantly and count the number of packets sent and received. We measure packet loss for 20 s per rate before increasing the rate by roughly 1 kpkts/s. Figure 3 depicts the packet loss with per-node IRQ pinning. By pinning IRQs to all cores of the NUMA node the NIC belongs to, memory copies across NUMA nodes are minimized.

An interesting pattern emerges for long lines where a single core handles multiple IRQs. Before packet loss increases linearly with the packet rate, packet loss can be observed for some rates. For example, with 64 hosts packet loss occurs in the interval 17.3 kpkts/s to 50 kpkts/s, but between 50 kpkts/s to 55 kpkts/s, no loss occurs. This is caused by a combination of rescheduling interrupts and soft interrupt flood handling. Between 17.3 kpkts/s to 50 kpkts/s IRQs are processed regularly. Due to the load balancing mechanism, the IRQs are rescheduled to a different core causing a spike in latency and packet loss occurs. Suo et al. [25] confirms that when CPU utilization is imbalanced, the kernel reschedules IRQs to other cores, causing rescheduling interrupts. The mechanism for dealing with soft interrupt floods explains that the stable packet rate before increases linearly [26]. The system is overloaded when it spends more than 2 ms for ten repeats processing soft interrupts. The kernel process `ksoftirqd/n` is scheduled to clear the remaining soft interrupts. Due to `ksoftirqd/n` running with a non-real-time scheduling policy, it can be preempted by other processes [27] unlike the IRQ handlers running with a real-time scheduling policy and high priority. Additionally, the `ksoftirqd/n` processes all remaining softirqs of any kind, which can result in higher, less predictable latencies for network softirqs. Although no packets are lost, the latency becomes increasingly unpredictable due to the non-real-time scheduling policy, larger polling intervals, and scheduling latency. Conversely, the highest packet rates are in direct conflict with latency.

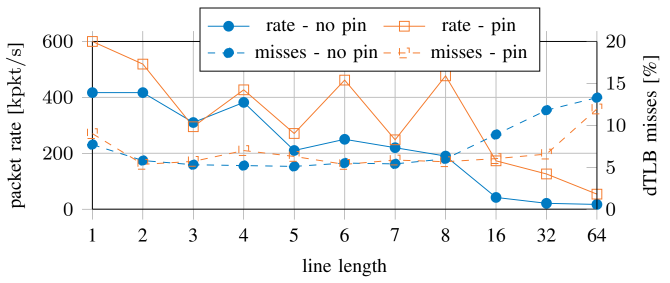

Figure 4. Packet rate, dTLB misses for lines with and without IRQ pinning

In the following, we relate the maximum achieved packet rate before packet loss occurs to the corresponding data translation lookaside buffer (dTLB) misses. Two variants are analyzed: pinning IRQs to all cores of the NIC's NUMA node (pin) or letting the scheduler decide (no pin). We allow the kernel to schedule IRQs on all cores except the first one; as it is already handling other interrupts from `irqaffinity=0`. To gather the data, we sample active IRQ threads for 10 s with `linux-perf` while the DuT is processing packets. The data presented in Figure 4 reveals that a higher packet rate is achieved by pinning IRQ threads to the correct NUMA nodes.

NUMA locality explains why lines with an even number of nodes achieve, in general, higher packet rates than uneven ones. The DuT, equipped with the first generation AMD EPYC, has 4 NUMA nodes. Ingress is attached to node 1, while egress is attached to node 3 as depicted by Figure 2. When using an uneven amount of containers, the IRQ handler of the egress must copy memory from node 1 to node 3. The delay added by the PCIe bus remains constant in both cases; using an even amount of containers bypasses the memory copy crossing NUMA domains at the final node in the line. In line with our data, Emmerich et al. [11] measured a penalty of 20 % on the packet rate for inadequate NUMA assignment.

With more nodes, the packet rate drops noticeably due to a higher number of dTLB cache misses, especially with 64 containers, with around 13 % dTLB misses. If a mapping between a physical and virtual page is absent in the dTLB, an expensive page walk stalls the current cycle due to multiple slow memory accesses. Two factors contribute to the increasing dTLB pressure: the high packet rate and, with more than 8 containers, overcommitment of cores handling IRQs. When utilizing 16 containers, the packet rate is measured at 173 kpkts/s, which is less than half of the rate achieved with 8 containers (477 kpkts/s) with per-node IRQ pinning. This performance degradation stems from each core processing effectively two IRQ handlers and inefficient scheduling.

None of the lines can saturate the maximum line rate of 10 Gbit/s divided by the number of used VFs per NIC. For example, a line with 64 containers with per node pinning can achieve throughput of 31 Mbit/s while the line rate per VF is at $\frac{10\,\text{Gbit/s}}{63} = 159\,\text{Mbit/s}$. Although SR-IOV incurs overhead, the OS and hardware bottleneck dominate the physical limitations.

B. Latency

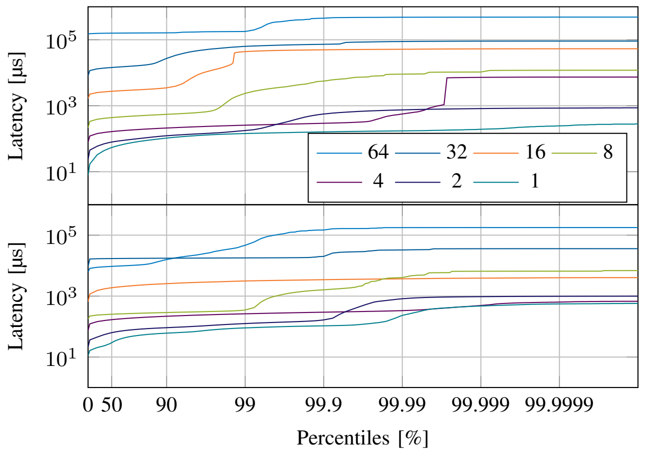

Figure 5. HDR diagrams of line lengths, max. packet rate; no IRQ pinning (top) and per node IRQ pin (bottom)

First, we measure with the maximum achievable packet rates as retrieved in Figure 4 to stress the system while avoiding overload, which would harm latency [21], [27]. Second, a packet rate of 17.3 kpkts/s as it is below the maximum packet rate for lines up to 64. Each experiment is repeated a minimum of three times, and the run with the highest worst-case latency is selected, remaining data can be accessed as described in Section VI. High-dynamic-range (HDR) diagrams visualize the latency distribution by highlighting the impact of outliers in a dataset with logarithmic scales on both axes.

Figure 5 shows the latency distribution of a line topology with selected lengths without IRQ pinning in the top diagram, and in the bottom with per NUMA node IRQ pinning at the respective maximum packet rate. Due to varying packet rates and our Intel X710 NICs batching aggressively on lower packet rates [21], the latencies in the two HDR diagrams are not comparable. Without IRQ pinning, the tail latency is less predictable, exhibiting frequent spikes. For example, at the 99.9^{th} percentile, a line with four nodes outperforms a line with two nodes. Furthermore, past the 99.99^{th} percentile, a sharp increase in latency is observed for four nodes, indicating that an interrupt was scheduled on the same core as the IRQ handler. When comparing the two HDR diagrams, it is evident that IRQ pinning reduces latency and improves predictability. While the lower percentiles exhibit similar values, with IRQ pinning showing slightly lower latencies, the tail latency diverges past the 99.9^{th} percentile. For instance, the tail-latency of a four-nodes-line is at 7400 µs without and 678 µs with IRQ pinning. With more than 8 nodes, the effect of cores processing more than one IRQ handler becomes apparent. Even at the median, the latency of a 16-nodes-line with 1094 µs diverges from the linearity of an 8-nodes-line at 335 µs.

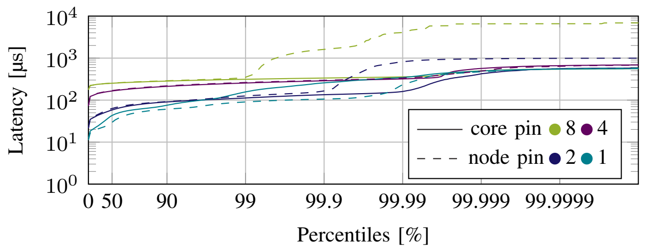

Figure 6. HDR diagram of line lengths, max. packet rate, node vs. core pin

To further investigate the impact of IRQ pinning, Figure 6 compares the tail latency of IRQ pinning per NUMA node and core. When pinning each IRQ handler to a single CPU core located on the same NUMA node, we are limited by the AMD EPYC 7551P to lines of length 8. Seven cores are necessary for the VFs on `port1` and `port2` (cf. Figure 2), and one core is required each for the ingress and egress. In scenarios where the amount of containers in a chain is less than the number of per-NUMA available CPU cores, per core IRQ pinning demonstrates stable, lower tail latencies. While up to the 99^{th} percentile, the latency is identical, rare rescheduling interrupts cause the tail latency to diverge when pinning IRQ handlers per NUMA node. A significant improvement at percentiles past the 99.9^{th} can be observed for 8 nodes. When comparing lengths one and two, 99.99 % of the recorded events are relatively close, with diverging tail latency past the 99.99^{th} percentile. For only one container, the kernel forwards packets between two NICs; however, for two containers, all four ports of both NICs are processing packets with SR-IOV, introducing additional delay. Concluding, IRQ pinning per core is strictly superior to per NUMA node.

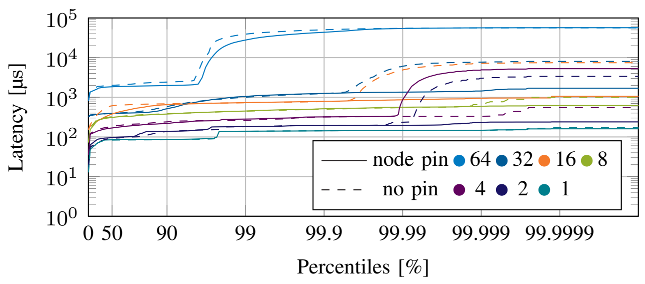

Figure 7. HDR diagram of line lengths, 17.3 kpkts/s

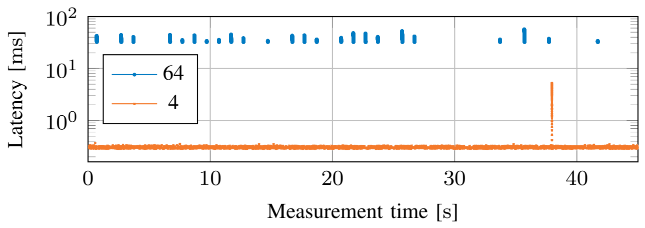

Figure 8. 5000 worst-case latencies, 17.3 kpkts/s, per-node IRQ pin

The latency distribution for a packet rate of 17.3 kpkts/s is shown in Figure 7. Most events occur between 100 µs to 1000 µs with latency increasing with number of nodes. NIC batching does not inhibit comparability, as the same packet rate is used for all measurements. There is no significant difference between the two variants. Between the 90^{th} and 99^{th} percentile, the latency of a 64-nodes-line is higher without IRQ pinning, converging with higher percentiles.

Most noteworthy are the large spikes in latency that occur infrequently for both variants. For example, with node pinning, a significant spike in latency to 5.2 ms for 4 nodes and 60 ms for 64 nodes can be observed. Experiments with the maximum packet rate as shown in Figure 5 do not exhibit such spikes. To further investigate this phenomenon, we display the 5000 worst-case latencies over time for lines with outliers and node pinning in Figure 8. One spike can be found at 38 s for 4 nodes caused by a rescheduling interrupt. The same applies to the spikes displayed for 64 nodes. The kernel scheduler balances processes across all available cores resulting in more rescheduling opportunities for a higher number of containers and therefore IRQs. We did not consistently observe these spikes in the remaining runs of the experiment for 1-32 nodes. For example, the other five repeats with four nodes did not show a spike, with 64 nodes it occurred consistently in all repeats. Although the container system with IRQ pinning achieved a nearly loss-free maximum rate of 64 kpkts/s, when lowering the rate to 17.3 kpkts/s occasionally packet loss occurs as shown in Figure 3. As Figure 8 displays, 64-nodes-lines are more prone to rescheduling interrupts. Each core is overcommitted by almost 8 times, leading to increasing dTLB misses and worsening tail latency past the 90^{th} percentile. This shows that the networking subsystem has no threading bottlenecks and can be used in multiple containers simultaneously.

We recorded higher latency than related work [6] for a single host. While we measured 49 µs at the median, they measured 13 µs. This disparity can be attributed to their use of PCI passthrough of ingress and egress, whereas we choose SR-IOV for flow injection. Furthermore, for their measurement, they used a simple `tc`-based forwarder, which copies packets on layer two between interfaces, bypassing the complexity of kernel processing with routing policies.

In summary, we found that pinning IRQs to cores rewards lower tail latencies than per NUMA node. In scenarios where more containers are deployed in a line than CPU cores per NUMA node are available, overcommitting cores is an option when the packet rate is low. For higher packet rates, dTLB misses degrade the system performance, and interrupts can cause more likely spikes for higher rates.

V. Recommendations

Containers are a more performative alternative to VMs for non-critical applications with relaxed real-time requirements. We observed lower latencies for containers than VMs. However, hardware choices must be carefully considered as NUMA locality significantly impacts the system's performance with the upper bound set by the NIC's associated NUMA node's CPU cores. Latencies are most stable when pinning IRQs to dedicated cores, followed by pinning IRQs to the NUMA node of the NIC as the second-best option with latency spikes due to rescheduling interrupts.

We cannot recommend using multiple containers for applications with hard real-time requirements. Our experiments demonstrated that containers cannot fulfill the strictest 5G URLLC requirement of 0.5 ms with the kernel networking stack. It is impossible to sufficiently isolate CPU cores from the kernel to avoid interrupts, confirming the assessment by Gallenmüller et al. [28].

VI. Reproducibility

To reproduce our results, we provide scripts, raw data, and all figures, including repeats, that are not in the paper due to space constraints at our accompanying website[1].

VII. Limitations

Hardware availability limited our experiments to a single setup. Future work includes experiments on monolithic Intel CPUs and AMD CPUs with faster interconnects.

In our examination, we focused on packet forwarding at the kernel level, as introducing a more complex application would introduce extra latency due to transferring memory to user space. Moreover, additional delays would be introduced when using cores from another NUMA node. This issue primarily affects lightweight applications with notable networking delays. We focused on a single flow and line topology scenario, whereas future work should include more complex, multi-flow scenarios as performed for VMs by Wiedner et. al. [1].

VIII. Conclusion

Our work demonstrated that containers are a viable alternative to VMs for real-time networking when using the flexible kernel networking stack. Our results show that reducing the impact of concurrent containers on each other is possible when using careful optimizations. Network emulators can benefit from containers' flexibility and scalability while maintaining low latencies. Various solutions implementing containers like Containernet already exist. However, they typically rely on kernel-based virtual networking, negatively impacting tail latencies [1]. Using real hardware and scaling with SR-IOV significantly improves latency. We have demonstrated that our container approach can achieve stable, low latencies.

[1]https://tumi8.github.io/applicability-hwsupported-containers

In this work, we measured the packet rate of containers in a line topology with varying lengths of up to 64 nodes. NUMA locality is crucial for achieving high packet rates, so we recommend pinning IRQ handlers to CPU cores on the NIC's NUMA node. We identified rescheduling interrupts as a significant factor for higher tail latencies. They can be mitigated by pinning IRQ handlers to the respective CPU cores the NIC is attached to limiting the line length to the number of available cores on the NIC's NUMA node. When using long lines with 64 nodes, dTLB misses increase to around 12 %, degrading the packet rate and tail latency.

We found that pinning IRQs to the NUMA node is necessary to achieve a stable packet rate without packet loss, as the scheduler is unaware of the NUMA topology. This shows that optimizing the pinning of IRQs is technical important to plan ahead to achieve the lowest stable latencies on arbitrary line lengths. Furthermore, this optimizations enhance real-world applications with a reduced network-induced latency and provide more possibilities to use processing applications within the boundaries of latency requirements.

In the future, we plan to evaluate user-space networking, such as DPDK, in concurrent containers. Previous work [6] suggests containers with DPDK can achieve lower latencies than with kernel networking reducing the kernels impact. By using DPDK, packets are processed in user space so that additional isolation with cgroups is possible to mitigate interrupts. Additionally, we intend to evaluate how different kernels and schedulers affect the tail latency. Moreover, the potential impact of newer Linux kernel versions on latencies must be analyzed. More complex topologies, more complex real world applications, and a flow setup to analyze the influence of different paths per flow is part of our future work.

Acknowledgements

This work was supported by the EU's Horizon 2020 programme as part of the projects SLICES-PP (10107977) and GreenDIGIT (4101131207), and by the German Federal Ministry of Education and Research (BMBF) under the projects 6Glife (16KISK002) and 6G-ANNA (16KISK107).

References

[1] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "HVNet: Hardware-Assisted virtual networking on a single physical host," in *IEEE INFOCOM WKSHPS: Computer and Networking Experimental Research using Testbeds (CNERT 2022)*, Virtual Event, May 2022.

[2] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering*. IEEE, Mar. 2015.

[3] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar. 2015.

[4] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and virtual machines at scale," in *Proceedings of the 17th International Middleware Conference*. ACM, Nov. 2016.

[5] D. Beserra, E. D. Moreno, P. T. Endo, and J. Barreto, "Performance evaluation of a lightweight virtualization solution for HPC I/O scenarios," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, Oct. 2016.

[6] F. Wiedner, M. Helm, A. Daichendt, J. Andre, and G. Carle, "Containing Low Tail-Latencies in Packet Processing Using Lightweight Virtualization," in *35rd International Teletraffic Congress (ITC-35)*, Oct. 2023.

[7] S. Huang and I. Baldine, "Performance Evaluation of 10GE NICs with SR-IOV Support: I/O Virtualization and Network Stack Optimizations," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 197–205.

[8] 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 Paralle and Distributed Processing (IPDPS)*. IEEE, 2010.

[9] N. Pitaev, M. Falkner, A. Leivadeas, and I. Lambadaris, "Characterizing the performance of concurrent virtualized network functions with OVS-DPDK, FD.IO VPP and SR-IOV," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. ACM, Mar. 2018.

[10] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471–1480, Nov. 2012.

[11] P. Emmerich, M. Pudelko, S. Bauer, and G. Carle, "User space network drivers," in *Proceedings of the Applied Networking Research Workshop*, ser. ANRW '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 91–93.

[12] J. Bainbridge and J. Maxwell, *Red Hat Enterprise Linux Network Performance Tuning Guide*, mar 2015.

[13] T. Herbert and W. de Bruijn. (2023) Scaling in the linux networking stack. Accessed on June 20, 2024. [Online]. Available: https://www.kernel.org/doc/html/latest/networking/scaling.html

[14] R. Rosen, *Linux Kernel Networking*. Apress, 2014.

[15] J. H. Salim, R. Olsson, and A. Kuznetsov, "Beyond softnet," in *Proceedings of the 5th Annual Linux Showcase & Conference*. Oakland, California, USA: USENIX Association, November 5–10 2001.

[16] A. Beifus, D. Raumer, P. Emmerich, T. M. Runge, F. Wohlfart, B. E. Wolfinger, and G. Carle, "A study of networking software induced latency," in *2015 International Conference and Workshops on Networked Systems (NetSys)*. IEEE, Mar. 2015.

[17] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. D. Rose, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, Feb. 2013.

[18] G. Ara, L. Lai, T. Cucinotta, L. Abeni, and C. Vitucci, "A Framework for Comparative Evaluation of High-Performance Virtualized Networking Mechanisms," in *Cloud Computing and Services Science*, D. Ferguson, C. Pahl, and M. Helfert, Eds. Cham: Springer International Publishing, 2021, pp. 59–83.

[19] M. S. Rathore, "KVM vs. LXC: Comparing performance and isolation of hardware-assisted virtual routers," *American Journal of Networks and Communications*, vol. 2, no. 4, p. 88, 2013.

[20] P. Emmerich, S. Gallenmüller, F. Wohlfart, D. Raumer, and G. Carle, "Moongen: A scriptable high-speed packet generator," *Proceedings of the 2015 Internet Measurement Conference*, 2014.

[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. 1, Dec. 2022.

[22] Intel Corporation, "E810 datasheet rev2.6," 05 2023. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/613875/intel-ethernet-controller-e810-datasheet.html

[23] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "Ducked Tails: Trimming the Tail Latency of(f) Packet Processing Systems," in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021, pp. 537–543.

[24] Advanced Micro Devices, Inc., *Software Optimization Guide for AMD Family 17h Models 30h and Greater Processors*, Advanced Micro Devices, Inc., March 11 2020. [Online]. Available: https://www.amd.com/system/files/TechDocs/56305.zip

[25] K. Suo, Y. Shi, A. Lee, and S. Baidya, "Characterizing networking performance and interrupt overhead of container overlay networks," in *Proceedings of the 2021 ACM Southeast Conference*. ACM, Apr. 2021.

[26] J. Corbet. (2016) Threadable NAPI polling, softirqs, and proper fixes. Accessed on June 20, 2024. [Online]. Available: https://lwn.net/Articles/687617/

[27] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications," *ArXiv*, vol. abs/1809.02595, 2018.

[28] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, “5G URLLC: A Case Study on Low-Latency Intrusion Prevention,” *IEEE Communications Magazine*, vol. 58, no. 10, pp. 35–41, 2020.