The State of Middleboxes Inside Containers

Josef Schönberger, Florian Wiedner*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: josef.schoenberger@tum.de, wiedner@net.in.tum.de

***Abstract*—Due to the reduced cost and increased flexibility, Network Function Virtualization (NFV) is a lucrative alternative for middleboxes compared to dedicated hardware. However, current NFV implementations are mostly VM-based despite the current trend toward containers for software deployments in the industry.**

This paper analyzes the current state of research on container-based NFV for middleboxes. We identify several issues that will need to be solved for widespread deployment, especially for low-latency applications. Finally, we discuss for which service level requirements container-based NFV might be viable for middleboxes.

***Index Terms*—containers, nfv, cnf, middleboxes**

1. Introduction

In recent years, *Network Function Virtualization* (NFV) has become a hot topic both in research and the industry. It promises to replace expensive dedicated hardware for specific network functionality with inexpensive off-the-shelf server hardware that provides these functions using software in virtual machines or containers. Especially for middleboxes, this presents the advantages of increased flexibility, scalability, updateability, and maintainability. So far, most implementations are based on system virtual machines.

At the same time, more and more software developers have started deploying their products in containers. Containers are a lightweight alternative to full-system virtual machines, which come with little overhead compared to bare-metal installations while providing a degree of isolation that is only slightly below virtual machines. Combined with automated container orchestration software such as Kubernetes[1], they offer automated scaling depending on the load and automated failover should one container fail. Finally, because these containers all run on a single kernel, no static resource allocation, such as CPU pinning, is necessary for containers, which allows for overprovisioning of resources, i.e., for more services per machine as long as the combined load of all services remains within limits.

While VM-based NFV has proven to be practical, using containers instead of virtual machines might also be lucrative for middlebox operators. However, this presents new challenges, as service level agreements might define strict limits on metrics like bandwidth, latency, reliability, and jitter. For example, the 5G *ultra-reliable low-latency communication* (URLLC) service category specifies an overall round-trip time of at most 1 ms and a reliability of at least 99.999% [1]. This requires a reevaluation of the viability and practicability of container-based network functions—also called *Cloud-Native Network Functions* (CNFs)—for middleboxes.

In this paper, we first provide an overview of some related work on this topic. We then discuss several potential issues, study the current state of research on them, and analyze the proposed solutions. We also offer potential approaches for solving some of these issues. Finally, we conclude with a discussion on whether containers might be viable for middleboxes.

2. Background

In this section, we define several terms that we will use throughout this paper.

2.1. Middleboxes

The term "*middlebox*" refers to all packet processing units that provide functionality beyond basic routing. Typical examples of middleboxes are firewalls, intrusion detection and prevention systems, and *Network Address Translations* (NATs).

2.2. Network Function Virtualization

The classical approach to middleboxes was using dedicated hardware for each function. Due to their low-volume nature, these devices are very costly. Furthermore, they are inflexible, with no possibility of customizing their behavior. Additionally, it is often impossible to install updates; and even if updates are possible, they have to be provided by the hardware vendor.

NFV solves these problems. It typically comprises virtual machines running on regular off-the-shelf server hardware, which can provide the same functionality in software [2]. These typically only offer simple functions and are composed together (in a process called "*Service Chaining*") to provide complex functionality.

Apart from solving the aforementioned problems with dedicated hardware, NFV also offers a few additional advantages. As the functions are implemented in simple virtual machines, they can be migrated from one server to another, e.g., for maintenance. They can also be replicated to dynamically adjust to increased load or changes in service level agreements. Finally, multiple *Virtual Network Functions* (VNF) can be run per machine, further reducing the overall hardware cost.

1. https://kubernetes.io/

 doi: 10.2313/NET-2024-09-1_05

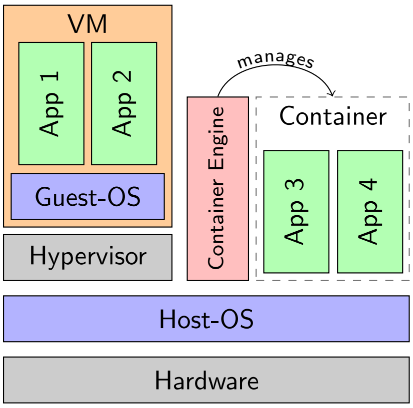

Figure 1: Comparison between VMs and Containers. Containers bundle processes and isolate resources such as the file system at the Host-OS level, thereby removing the Hypervisor and Guest-OS overhead.

2.3. Containers

Most NFV solutions are currently based on system virtual machines. Containers are a lightweight alternative; their architecture is shown in Figure 1. Instead of virtualizing hardware components and resources, they rely on mechanisms of the underlying operating system's kernel to provide an isolated view of system resources. This results in a more efficient architecture in which neither a guest operating system nor a hypervisor is required, but processes exist directly on the host operating system instead. These are bundled into groups called "containers" with a common view on resources such as the file system (and, thereby, system libraries) and network devices. The container engine (which sets up, monitors, and destroys the containers) and isolation mechanisms themselves introduce very little overhead [3] and are comparatively cheap to create and destroy [4].

Due to their architecture, resource allocation for containers is far simpler than for regular virtual machines. For example, containers do not generally need to statically allocate CPUs, memory, or storage [5].

However, the overall attack surface is increased since all containers operate on a common host operating system. A vulnerability in the host kernel, such as a privilege escalation, can be used to compromise the kernel not only for the malicious container but also for all containers and services on the system [6].

3. Related Work

Several studies identify hindering factors for commercial deployments of CNFs for middleboxes [7]. This paper focuses on container-based network functions specifically and will therefore not discuss the general problems of NFV for middleboxes.

Some previous work has already identified and discussed several problems. This section overviews attempts to develop complete CNF platforms that can be used for middleboxes. We will later discuss relevant literature concerning each issue in the relevant sections.

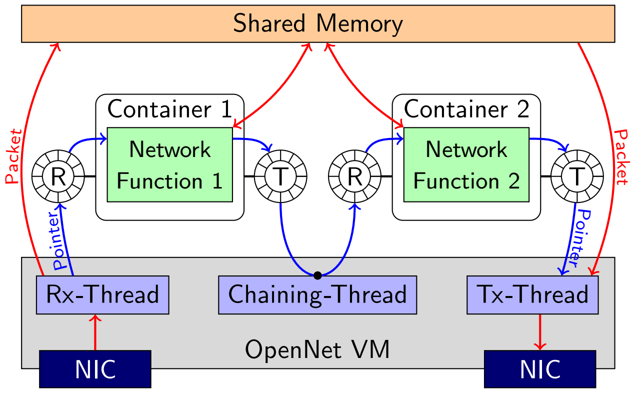

Figure 2: Architecture of OpenNetVM [8].

In 2016, Zhang et al. [8] presented *OpenNetVM*[2]. It executes the CNFs in separate Docker containers. The architecture is shown in Figure 2. Packets are exchanged in a shared memory region to avoid repeatedly copying packet data. Only the addresses of the packets within the shared memory region are transferred; the network functions read and write them in ring buffers and separate chaining threads on the host forward them from one network function to the next. Rx- and Tx-Threads transfer the packets between the shared memory and the *Network Interface Cards* (NICs). OpenNetVM uses the *Data Plane Development Kit*[3] (DPDK), a high-performance userspace network driver and application framework, for communication with the NICs.

Zheng et al. [9] implemented *MVMP*, which extends OpenNetVM's architecture by sharing a ring buffer between chained functions to eliminate the chaining threads. In their evaluation with a simple forwarding application, they measure a near-identical throughput to a reference DPDK application that runs on the host system directly.

Dzeparoska et al. [10] made an effort to develop CNFs for *Cloudify*, "an open source cloud orchestration framework" [11]. As opposed to [8] and [9], their VNFs exclusively operate in an *Software Defined Networking* (SDN) environment based on *OpenStack*[4] and interact with the network using Linux bridges and virtual ethernet devices. Dzeparoska et al. [10] compared the latency of their container-based solution to Cloudify's existing VM-based solution and consistently measured a significantly lower latency for containers.

4. Potential Issues with CNF Middleboxes

In this section, we identify several potential problems that might hinder the widespread usage of CNFs for middleboxes and discuss their current state in research.

4.1. Lack of Data for Realistic Scenarios

Plenty of literature exists regarding container-based network functions and their performance advantages compared to VM-based NFV. Some studies develop and evaluate real-world applications with multiple chained VNFs

2. https://sdnfv.github.io/onvm/
3. https://www.dpdk.org/
4. https://www.openstack.org/

 doi: 10.2313/NET-2024-09-1_05

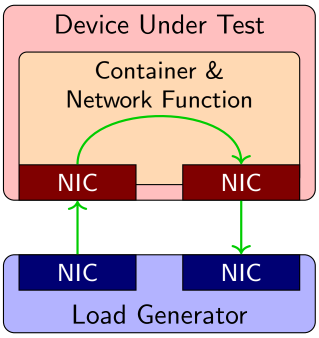

Figure 3: Typical evaluation setup with forwarding function.

with promising results [12]. However, most of them do not study realistic middlebox scenarios. For example, much work focuses on VNFs in SDN that are part of a control plane, which do not have the same bandwidth and latency requirements as middlebox VNFs on the data plane. While some have analyzed the performance of middlebox applications (such as intrusion detection systems) in particular, their experiments' deployment scenarios are still unrealistic.

In the end, there are several aspects that realistic experiments should consider:

4.1.1. Insufficient System Load. Many studies on container-based network functions analyze their performance with simple forwarding or acknowledging applications, shown in Figure 3. While these are great for reducing noise in the measured data and isolating the performance overhead of the container environment, they do not provide realistic scenarios, as resource contention might remain an unidentified problem. For example, none of the studies mentioned in Section 3 use realistic resource-intensive workloads [8]–[10]. As a result, the true impact of a high system load on the bandwidth and especially the latency is still unknown, particularly with high CPU, memory, and/or I/O bandwidth usage.

This becomes especially problematic for scenarios with multiple (possibly chained) VNFs per host: In 2023, Attaoui et al. [13] conducted a study on existing work regarding the placement of VNFs. In section IV subsection A specifically, they mention that "containers fight for the same resources in the system" [13]. Because existing container engines do not provide enough fairness guarantees for resource distribution, this can result in contentions on system resources. The authors also explicitly recommend using multiple virtual machines and strategically distributing VNF containers within those to alleviate the problem despite the reintroduction of additional virtualization overhead.

4.1.2. "Clean" Traffic. For most publications on CNFs, the performance evaluation is based on "clean" traffic: the packets were of constant size, perfectly paced (i.e., spread equally on the timeline), and had no short bursts. When handling real-world traffic, the VNFs are expected to deal with imperfect traffic without significant performance implications. Yet, the behavior of container-based VNFs in these scenarios has yet to be analyzed.

4.1.3. Insufficient Data on (Tail) Latencies. Modern systems can have stringent requirements on (tail) latencies. For example, the 5G URLLC scenario specifies an overall end-to-end round-trip time limit of as little as 1 ms [1]. Combined with the high-reliability requirement of 99.999% [1], this imposes significant challenges for VNFs on the 5G URLLC data plane.

All publications mentioned in Section 3 do not analyze latency at high-percentile tails. Additionally, most publications on CNFs only analyze the average latency, if they analyze the latency at all.

Only a few publications directly address very low tail latency requirements. Gallenmüller et al. [14], [15] analyzed the difficulties in archiving low high-percentile tail latencies for bare-metal network functions on Linux. Gallenmüller et al. [16] also successfully reduced the latency of a real-world application (in particular, the intrusion prevention system *Snort 3*[5]) significantly. While their work proves that Linux can generally be a suitable platform for low-latency network functions, several deployment obstacles must be overcome. Most notably, a specific realtime-optimized Linux kernel with several kernel parameters and CPU pinning (i.e., reserving a specific CPU core and restricting a task to it) is required.

Wiedner et al. [17] extended this work to containers and proved that CNFs are generally a viable option for very low (tail) latency applications. Wiedner et al. [18] also analyzed the impact of cgroups v2 (a fundamental building block for Linux containers responsible for resource isolation) on tail latencies in container-based VNFs. However, these publications also only evaluate with a single forwarding network function.

In summary, while it has been shown that very low tail latencies are possible in principle, practical container network function implementations for these scenarios have yet to be developed.

4.1.4. Container Interferences. Multiple container-based VNFs on a single host might interfere with each other. For example, as previously mentioned, containers fighting for a common set of resources can decrease performance [13]. The effects of these interferences have yet to be analyzed.

One potential problem stems from synchronization. Since most experiments only use simple forwarding functions for evaluation, the impact of thread synchronization is never measured. However, since all threads operate on the host kernel directly (c.f. Figure 1), all thread synchronization is centered on it. This may have implications on the (tail) latency for middlebox VNFs that extensively use thread synchronization, e.g., for concurrent accesses on data structures.

A similar problem arises from in-kernel contention. In VM-based NFV, many kernel tasks, such as memory management, are distributed across all VNFs. In contrast, for container-based network functions, it is plausible that in-kernel contention (e.g., on the memory allocator or scheduler) could further increase the (tail) latencies.

Finally, TLB shootdowns cause latency spikes [14], [15], [19]. Each core has an independent *Translation Lookaside Buffer* (TLB), which effectively caches address

5. https://www.snort.org/snort3

 doi: 10.2313/NET-2024-09-1_05

translations. Since the CPU does not enforce TLB consistency, modifications or invalidations of address mappings are not immediately visible to all cores. Therefore, the issuing processor must send a so-called TLB shootdown—a broadcasted *Inter Processor Interrupt* (IPI) that disrupts all other CPUs' execution in order to flush all TLBs. On VM-based NFV, these TLB shootdowns are restricted within the guest operating system and the virtual machine; in contrast, on container-based network functions, they are broadcasted to all cores and thereby disrupt all VNFs.

4.2. Resource Contention from Overprovisioning

VM-based NFV naturally does not suffer from resource contention due to static resource allocation; since the hypervisor typically assigns each VM a (mainly) static share of CPUs, memory, and I/O devices, different VNFs are independent regarding resource distribution. However, one of the major advantages of containers is that static resource allocation similar to VMs is not necessary—the host operating system allocates the individual resources on demand. But this can cause resource contention in container-based network functions with a significant performance impact [13]. Additionally, Wiedner et al. [17] and Gallenmüller et al. [14], [15] have shown that CPU pinning is essential for low tail latencies.

This is directly at odds with another expectation on containers: the ability to overprovision the system resources such as CPU time and memory. In this context, overprovisioning means creating more containers than the system resources allow under full load. In other words, if all containers were to use all of their available resources simultaneously, the operating system would not be able to fulfill the requirements of all containers. This can make sense for containers as it is rare that all containers simultaneously require all resources and because it is possible to migrate and thereby offload VNFs to other hosts in cases of high resource pressure [5].

Additionally, resource overprovisioning is essential for self-healing container replication as implemented in Kubernetes, for example. Spawning multiple instances of the same VNF not only allows for dynamic scaling based on load but also allows the orchestrator to spawn fallback containers that do not yet accept any work but are ready to take over at any point should another active container fail. Such backup containers are idle for most of their lifetime and would thereby waste resources under static resource allocation.

In the end, there is a lack of an analysis of the implications of resource overprovisioning beyond the studies that reveal general problems with resource contentions in CNFs. However, data on this matter would be critical since software developers should be able to decide whether static resource allocation is unavoidable under a given set of service level requirements.

4.3. Latency Spikes at CPU Migration

CPU migration (i.e., the kernel moving a task from one core to another) needs to temporarily stop the execution of the VNF, which causes a latency spike. The new core also does not have the data ready in its caches, further negatively impacting performance. For this reason, Gallenmüller et al. [14], [15] recommend isolating the container threads to a single core. However, this effectively results in static resource distribution for CPU cores, as this core is now reserved for this single thread, which hinders overprovisioning. As a result, newer approaches to thread migration are necessary for low-latency applications.

We propose that this problem is solvable. Instead of leaving the migration to the kernel's dispatcher, CPU core migration should only be done by replacing the application thread. In other words, the application should spawn a new worker thread on the new core, request the existing threads to stop accepting packets, and finally terminate the previous threads.

Of course, this approach also has drawbacks. The new application has not yet established its memory working set, so the caches are not filled with relevant data. Similarly, on NUMA systems, moving the relevant data from one node to the other might be necessary, which is difficult considering that the thread on the previous node might still actively use this data. Additionally, creating processes and threads involves TLB shootdowns on all cores, which could result in latency spikes in other independent VNFs. Finally, this approach requires both the old and new cores to be allocatable at the same time. As a result, if a system wants to remain able to move a thread from one core to another, it always needs to keep a spare core unoccupied.

In the end, this problem will need to be addressed by future work.

4.4. Service Chaining

Depending on the service level requirements, different approaches to service chaining might be necessary. Most research on service chaining of CNFs focuses on throughput [20], with some measuring the average latency as well [21].

If the overall end-to-end latency of the complete chain is not critical, regular approaches using default Linux networking mechanisms such as bridges and virtual ethernet (`veth`) devices [20] or SDN mechanisms such as *Open vSwitch*[6] might be sufficient [10], [21] and potentially preferable due to their simplicity and flexibility. On the other hand, if high throughput and low (tail) latency are essential, more sophisticated approaches based on shared memory might be necessary [8], [9]. The current approaches could probably still be optimized by further reducing the amount of threads touching the packets, e.g., by placing NICs into the container directly using *Single Root I/O-Virtualization* (SR-IOV), which could, for ,example eliminate the dedicated Rx- and Tx-Threads in OpenNetVM and thereby potentially lead to improved cache locality. Additionally, the existing approaches are still comparatively inflexible.

5. Conclusion

There is no scientific consensus on whether CNF-based middleboxes are viable. While several papers successfully implemented CNFs [8]–[10], others are more

6. https://www.openvswitch.org/

skeptical and actively encourage the use of VMs instead [13], [16]. We showed that there are several aspects in which we lack sufficient data to come to a conclusive answer. In particular, the behavior with real-world applications and data, tail latencies in realistic scenarios, and interferences of concurrent CNFs are still largely unknown. We also demonstrated that resource overprovisioning can cause resource contention and that a VNF CPU migration could result in latency spikes, for which we proposed a new approach to reduce the impact. Finally, we also showed that while service chaining for CNFs has been explored, we believe there to be room for improvement.

Ultimately, we also conclude that the suitability of CNFs for middleboxes heavily depends on the service level requirements. Assuming that the security benefits of virtual machines over containers are negligible for the use case, if only high bandwidth and efficiency are required, CNFs are well-tested and a good choice. If a low average latency is specified, container-based middlebox VNFs could still be viable. However, if very low tail latencies should be guaranteed, several pitfalls remain. Nevertheless, we believe that all of the aforementioned problems can be solved, even though much future work is still required.

References

[1] European Telecommunications Standards Institute (ETSI), "Study on scenarios and requirements for next generation access technologies," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.913, 08 2017, version 14.3.0.

[2] ——, "Network functions virtualisation – an introduction, benefits, enablers, challenges & call for action," *SDN and OpenFlow World Congress*, Oct. 2012, accessed on 2024-04-07. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[3] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 342–346.

[4] R.-S. Schmoll, T. Fischer, H. Salah, and F. H. P. Fitzek, "Comparing and evaluating application-specific boot times of virtualized instances," in *2019 IEEE 2nd 5G World Forum (5GWF)*, 2019, pp. 602–606.

[5] S. K. Tesfatsion, C. Klein, and J. Tordsson, "Virtualization techniques compared: Performance, resource, and power usage overheads in clouds," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 145–156. [Online]. Available: https://doi.org/10.1145/3184407.3184414

[6] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52 976–52 996, 2019.

[7] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "Network functions virtualization: The long road to commercial deployments," *IEEE Access*, vol. 7, pp. 60 439–60 464, 2019.

[8] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood, "OpenNetVM: A platform for high performance network service chains," in *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, ser. HotMIddlebox '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 26–31. [Online]. Available: https://doi.org/10.1145/2940147.2940155

[9] C. Zheng, Q. Lu, J. Li, Q. Liu, and B. Fang, "A flexible and efficient container-based NFV platform for middlebox networking," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 989–995. [Online]. Available: https://doi.org/10.1145/3167132.3167240

[10] K. Dzeparoska, A. Nasiri, and B. Najafi, "Deploying container-based NFV within SDN environment," 03 2017, accessed on 2024-04-04. [Online]. Available: https://doi.org/10.13140/RG.2.2.29961.88166

[11] Cloudify Documentation Center – About Cloudify. Accessed on 2024-04-05. [Online]. Available: https://docs.cloudify.co/latest/about/

[12] D. T. Nguyen, N. L. Dao, V. T. Tran, K. T. Lang, T. T. Pham, P. H. Nguyen, C. D. Pham, T. A. Pham, D. H. Nguyen, and H. T. Nguyen, "Enhancing CNF performance for 5G core network using SR-IOV in Kubernetes," in *2022 24th International Conference on Advanced Communication Technology (ICACT)*, 2022, pp. 501–506.

[13] W. Attaoui, E. Sabir, H. Elbiaze, and M. Guizani, "VNF and CNF placement in 5G: Recent advances and future trends," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4698–4733, 2023.

[14] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "Ducked Tails: Trimming the tail latency of(f) packet processing systems," in *3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking (HiPNet 2021)*, Izmir, Turkey, Oct. 2021.

[15] 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. [Online]. Available: https://doi.org/10.1007/s10922-022-09710-3

[16] 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, Oct. 2020.

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

[18] F. Wiedner, A. Daichendt, J. Andre, and G. Carle, "Control Groups Added Latency in NFVs: An Update Needed?" in *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2023.

[19] E. Rigtorp. (2020, Mar.) Low latency tuning guide. Accessed on 2024-04-02. [Online]. Available: https://rigtorp.se/low-latency-guide/

[20] S. Livi, Q. Jacquemart, D. L. Pacheco, and G. Urvoy-Keller, "Container-based service chaining: A performance perspective," in *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, 2016, pp. 176–181.

[21] R. Cziva, S. Jouet, K. J. S. White, and D. P. Pezaros, "Container-based network function virtualization for software-defined networks," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, 2015, pp. 415–420.

 doi: 10.2313/NET-2024-09-1_05