Supporting Hybrid Virtualization Orchestration for Edge Computing

Giovanni Bartolomeo
TU Munich
Germany
giovanni.bartolomeo@tum.de

Patrick Sabanic
TU Munich
Germany
patrick.sabanic@tum.de

Nitinder Mohan
TU Delft
The Netherlands
n.mohan@tudelft.nl

Jörg Ott
TU Munich
Germany
ott@in.tum.de

Abstract

Microservice architectures allow developers to decompose their applications into independently deployable functional blocks, each with its own requirements. In order to support a wide range of constraints, service virtualization can be customized across microservices but is typically homogeneous within a cluster. As there is no clear *one size fit all* approach, we can improve resource utilization and performance by using virtualization as a new dimension in orchestration, especially in edge computing environments. For instance, Unikernels represent a lightweight virtualization technology that offers a performant alternative to traditional containers. While we find different studies analyzing and comparing these virtualization technologies, (a) the performance results might vary when including the overhead of the orchestration platform, and (b) it's not trivial to select the perfect virtualization technology for an entire cluster. In this paper, we explore the benefits of hybrid container-unikernel deployments by extending an orchestration framework for edge computing to allow for seamless mixing and matching of both technologies. Our evaluation shows how hybrid deployments can lead up to 44% CPU reduction cluster-wide while there are scenarios where containers are still preferable.

CCS Concepts

• **Computer systems organization** → **Cloud computing**; • **Information systems** → *Information systems applications.*

ACM Reference Format:
Giovanni Bartolomeo, Patrick Sabanic, Nitinder Mohan, and Jörg Ott. 2025. Supporting Hybrid Virtualization Orchestration for Edge Computing. In *The 8th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3721888.3722093

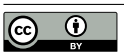
1 Introduction

In the landscape of modern computing, microservice architectures have increasingly become the standard approach for designing highly scalable and available applications. Microservices are typically deployed in containers [13, 28, 54]. With the OCI standardization [7], containers are nowadays decoupled from the underlying runtime, allowing for seamless portability across different environments and reusability of common functions – like nginx web servers, Redis, etc. This calls for a new dimension in orchestration, where the virtualization technology can be chosen based on the service requirements. Unfortunately, state-of-the-art still considers virtualization coupled with infrastructure provisioning. Edge computing significantly alters this assumption, given its inherently heterogeneous infrastructure offering variations in (CPU/memory) hardware, OS support, etc. [39, 45]. Previous studies highlight the operational overheads caused by cloud-native assumptions at the edge [15, 41] and benefits of lightweight virtualization in conjunction with containers [21, 29]. Unikernels are a good candidate for the edge because of their small footprint, faster instantiation, improved performance, and flexibility [35, 53]. However, despite advancements in unikernel toolchains, such as Unikraft [31], which allows porting existing Linux applications, the ecosystem does not support a wide range of applications and driver functionality [25]. Moreover, as shown in the remainder of this paper, *unikernels are not the best choice for all services at all times.* We envision a future where, given a standardized packaging format like OCI, the runtime can be chosen dynamically based on application requirements, with the orchestration platform effectively becoming a middleware for multi-virtualization setups. Take, for example, a stream-processing video analytics pipeline, which can include several GPU-intensive services that operate more suitably as containers with full-fledged OS providing complex driver support [55]. However, services within the pipeline, such as load balancers, may be more performant as unikernels using a hypervisor as resource multiplexer [29, 33, 36]. Hybrid virtualization also enables a gradual transition of complex containerized applications to unikernels – as the build toolchain evolves to support more system calls and libraries [1, 6, 23]. While several papers have empirically evaluated and compared the performance of different isolation technologies [29, 33, 49], they do not consider (i) the overheads of compatibility layers which allow these virtualizations to operate on common hardware and (ii) orchestration overhead for managing deployments with different virtualizations at runtime.

This paper explores the feasibility of container-unikernel hybrid orchestration. Our contributions are as follows.

(1) We extend Oakestra [15], a lightweight orchestration framework for edge computing. We implement a compatibility layer that allows Unikraft [31] unikernels to behave as containers from an orchestration perspective. We extend the control plane to aggregate clusters' virtualization information and the scheduling workflows to consider virtualization requirements. We introduce service *hotswap* to change the service's virtualization technology at runtime.

(2) We evaluate the suitability of hybrid virtualization orchestration via real-world application pipelines. Specifically, we dissect the overhead of the compatibility layer performing cross-deployment of



EdgeSys '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1559-4/2025/03
https://doi.org/10.1145/3721888.3722093

containers via `runc`, unikernels, and `gVisor` secure containers. Our results showcase the potential for hybrid virtualization, achieving up to ≈ 44% CPU usage reduction in our cluster.

2 Background and Related Work

Due to their small memory footprint (approx. a few MB) and reduced system call dependencies [32], unikernels offer faster boot times (≈ 10×) compared to containers [26, 31] and are easy to scale and migrate [37, 40, 51]. The shared application and kernel address space allows all code to run in the same CPU privilege domain, which improves performance by avoiding application and kernel context switches [52]. Early unikernel frameworks, such as MirageOS [4], required developers to write applications from scratch. However, in recent years, unikernels have evolved as a capable alternative to containers. Unikraft [31] provides a streamlined approach to building and porting existing Linux applications. The toolchain provides a high degree of POSIX compatibility (≈ 160+ out of 224 syscalls required for popular Linux install [34]). EVE-OS [1], a Linux Foundation project, is a universal, vendor-agnostic OS for edge computing hardware (including embedded devices) and adds native support for both containerized and unikernels workloads. Container runtimes (e.g., Firecracker [2], gVisor [3]) enhance container security and isolation by executing them as para-virtualized microVMs over `qemu/kvm` similar to unikernels. The OCI standards help supporting both container and unikernel runtimes simultaneously [7, 11]. Experimental runtimes like `urunc` [42] and `runu` [46] represent the first steps towards kernel-level compatibility unifying containers and unikernels [47]. Unfortunately, it is not clear what the overhead of such compatibility layers is in real-world deployments, and how they affect the orchestration of services, especially on constrained hardware at the edge. Edge infrastructure is generally less powerful and more heterogeneous than cloud datacenters, often comprising of smaller devices with varying CPU architectures and capabilities, e.g., Intel NUCs, Jetson Xavier, Raspberry Pis, etc. As edge computing is often seen as an extension of the cloud, the majority of orchestration solutions adapt the popular cloud-native Kubernetes (K8s) framework [19]. Solutions like KubeEdge [20], KubeFed [24], and MicroK8s [17] modify Kubernetes by simplifying control-plane operations and removing non-essential components to make it applicable for edge. On the other hand, Oakestra [12, 15] rearchitects the orchestration control plane from the ground up to address the hardware heterogeneity and geographical diversity in edge infrastructures with minimal overhead. In Oakestra, computational devices (*leaf* nodes) are grouped into (logical) clusters managed by local *cluster* orchestrators (see fig. 1). The worker node includes `NodeEngine` component for managing service deployment and operation and `NetManager` for network communication. Each *cluster orchestrator* is responsible for keeping track of fine-grained resource and service management within its cluster. The *root orchestrator* acts as an "orchestrator of clusters" and the point-of-contact of developers to deploy their applications.

Unfortunately, all state-of-the-art orchestration frameworks treat virtualization as a cluster constraint. We finally have an opportunity to exploit virtualization as a dimension to improve resource utilization and application performance. In [38], the authors examine approaches for orchestrating sandboxed containers as microVMs over `qemu/kvm` via extensions to K8s. FADES [21] leverages MirageOS [4] unikernels to deploy application microservices in Xen-bootable images suitable for edge devices. However, arguably (i) not all applications perform better as unikernels, and (ii) the virtualization technology must be dictated by application requirements and not by the infrastructure availability alone.

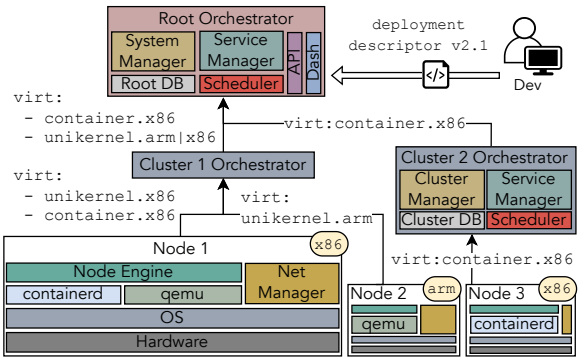


Figure 1: Oakestra with hybrid unikernel support.

3 Orchestration Support

In our exploration, we extend the Oakestra [12, 15] orchestration framework to support hybrid container-unikernel deployments and measure the overheads and benefits of such hybrid virtualization setups. We choose Oakestra due to its lightweight implementation and extensible design, which allows us to integrate unikernels orchestration metrics alongside containers with minimal changes and reduced overhead. We use Unikraft [31] as the unikernel runtime for our experiments, as it provides a wide range of unikernel configurations and supports a variety of applications. The proposed architecture provides an extensible interface that is used to evaluate unikernel virtualization as an additional orchestration dimension, but that easily allows for further runtimes support and optimal virtualization selection.

3.1 Hybrid Service Deployment

To enable hybrid virtualization support, we must ensure that the worker nodes' hardware can support container and unikernel execution. Oakestra supports integration of new runtimes via (a) using the runtime dispatcher interface or (b) integration to `containerd` thanks to OCI `runtime-spec` compatibility. Initial experiments with `runu` [46], an OCI-compatible runtime for `containerd`, showed inconsistent behavior. This runtime is currently under development [47], so misalignments with the latest Unikraft versions are expected. Moreover, managing `runu` as OCI runtime involves the additional overhead of the `containerd` middleware managing the hypervisor, which can be avoided by directly interacting with `qemu`. To overcome this, we design and implement a `Unikernel Runtime Abstractor` component in Oakestra's `NodeEngine`, which instead of controlling unikernels via `containerd`, adds abstractions for directly managing and monitoring Unikraft services within the orchestration framework (see fig. 2). This approach, while Oakestra-specific, is not replacing the OCI runtimes such as `runu`/`urunc`. These runtimes can be easily integrated as `containerd` runtimes when they mature, but at the cost of additional overhead. Unikernels (and the abstractor) are only enabled on machines supporting

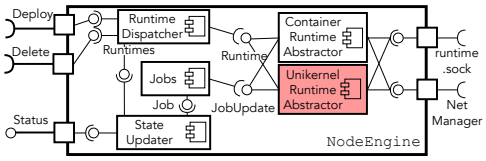


Figure 2: Unikernel support (in red) in Oakestra *node engine* component at the worker machine.

qemu and kvm targets (e.g., node 1 and 2 in fig. 1) while containers are enabled only in nodes supporting containerd. The Unikernel Runtime Abstractor (i) manages the lifecycle of unikernels and interacts directly with qemu for the virtualization, (ii) performs the downloading and unpacking of kernel images, and (iii) binds a routine to read the qemu qmp socket interface and update the internal service status (running/paused/failed) to the cluster orchestrator.

3.2 Resource Management and Scheduling

As shown in fig. 1, typical edge deployment with Oakestra may include worker nodes with heterogeneous hardware architecture and runtime target support. While some nodes execute both container and unikernel deployments (e.g., *node 1*), others only support either one of the two (e.g., *node 2* or *3*). At worker startup, the NodeEngine checks the nodes' CPU architecture and virtualization support. Then, it advertises its runtimes to the associated cluster orchestrator as virt tuple (<virtualization,architecture>). In Oakestra, the cluster orchestrator hides the internal infrastructure details of its workers by only sending the aggregated cluster statistics to the root orchestrator [14]. We extend the control plane to support multi-virtualization by propagating the set of all available virtualization and hardware architecture combinations in each cluster (see Cluster1 → Root in fig. 1). Further, we extend the Oakestra schedulers to consider virtualization during scheduling. Specifically, the *service schedulers*, before they apply one of the available scheduling policies [15], they first prune the most suitable cluster(s)/worker(s) list based on the available virtualization options matched with SLAs requirements.

3.3 Virtualization Hot-Swapping

We extend the control plane with a runtime switch functionality for stateless applications supporting multiple virtualization technologies. Suppose service1.instance1 is deployed as a container but a unikernel implementation is available. By triggering the hot-swap, the control plane performs the deployment of service1.instance2 unikernel alongside the first instance. The network component gradually balances and shifts the traffic from the container to the unikernel instance. Once the traffic migration is complete the first containerized instance is removed.

3.4 Inter-Service Networking

To achieve agile hybrid virtualization, it is important that the orchestrated services can interact with both unikernel and container-based services without additional overhead. Oakestra utilizes a semantic overlay network to enable multi-cluster container networking and load balancing. Each service is allocated IP addresses mapped to different load-balancing strategies across available instances. The NetManager interprets packets to/from a semantic address and re-assigns them to the correct instance IP address – forming a tunnel between communicating services. To achieve similar seamless networking between containers and unikernels (and across unikernels), we extend the NetManager to provision (i) a network namespace for unikernels and (ii) a local namespace IP address that can be used to translate network packets irrespective of the virtualization target. Unlike containers, unikernels do not share the host kernel but require a dedicated network stack. We overcome this by connecting a macvtap interface in bridge mode directly to the veth of the service's network namespace (see <s2 namespace>). The *runtime abstractor* is providing such interface to qemu for the unikernel startup. With this extension, unikernel and container namespace IP addresses are provisioned using the same Oakestra mechanism, giving out-of-the-box support for semantic load balancing and service discovery for both supported virtualization techniques.

4 Application Performance.

We evaluate hybrid virtualization orchestration for two application use cases. Firstly, we focus on a typical Edge/IoT sensor network application composed of commonly re-used services – web server, database, and message broker – and care for high throughput and availability (see §4.1). Our second application is a latency-critical augmented reality (AR) pipeline, which (structurally) is deemed as the killer application for edge computing [10] (see §4.2). For each use case we measure the applications' end-to-end latency and throughput, as well as the total system CPU consumption, including the application, the orchestration system, and compatibility layers usage. Note how memory usage is not reported as unikernel memory is fixed and depends on allocation at qemu startup. We conduct experiments on a heterogeneous hardware cluster inter-connected with 1 Gbps ethernet (< 1 ms RTT). The cluster includes (i) 2× servers *S1* with two AMD EPYC 7302 CPUs, two NVIDIA A40 GPUs, and 264 GB memory and *S2* equipped with Intel i9 CPU, two NVIDIA RTX 2080 GPUs, and 128 GB memory; (ii) *X1* 1× *Intel NUC* with Intel i5-6260U 4 core CPU and 16GB DDR4 memory; (iii) *X2* 1× APU with Intel i5-8400T 6 core CPU and 8GB DDR4 memory and (iv) *Pi* 2× *RPi 4* with ARMv8 Cortex-A72 4 core CPU and 8GB DDR4 memory. We install Oakestra components on all devices and orchestrate them as a single cluster to avoid inter-cluster scheduling overheads.

4.1 Hybrid Edge/IoT Application

To evaluate the impact of hybrid virtualization, we deploy a benchmark application that emulates the architecture of a real-world edge IoT sensor network commonly used within smart-factories [18] and smart-cities [30] (see fig. 7a). The distributed sensors collect data and forward them to the *nginx* proxy. The proxy load balances the data to the *analytics* service, which processes the data and forwards it to a pipeline of *key-value store*, *message broker*, and *web server* microservices for analytics and storage operations. We first benchmark the baseline container vs. unikernel performance for all three services and finally compose them into an end-to-end pipeline with two additional business logic services (*analytics* and *store manager*). We further evaluate the impact of hybrid virtualization on overall resource usage (including orchestration overhead) by *hot-swapping* the virtualization target within the pipeline.

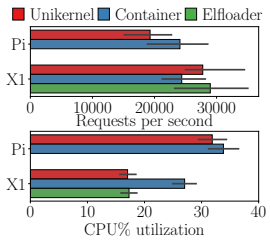


Figure 3: Redis.

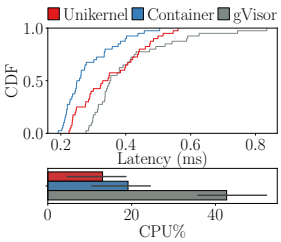


Figure 4: Nginx.

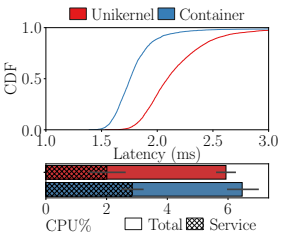


Figure 5: MQ Broker.

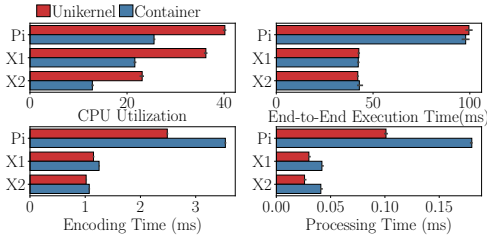
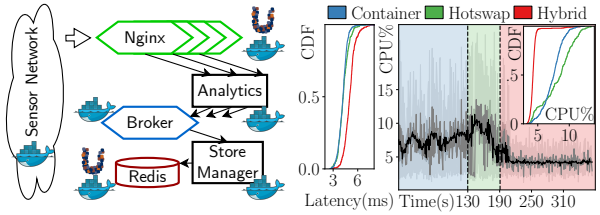


Figure 6: Augmented Reality Pipeline.



(a) Architecture. **(b) Performance.**

Figure 7: Edge/IoT Application.

4.1.1 ***Key-value datastore.*** Redis is a well-known in-memory key-value store. We used Unikraft's open-source implementation [8] and repackaged the same Redis binary as a Docker container to ensure similar functionality regardless of virtualization. Additionally, we also created a unikernel image from the native Redis binary using Unikraft `x86` `elfloader`. The `elfloader` allows running most of the traditional Linux executables in Unikraft. In the future, this method is likely to be employed for porting Linux applications without bespoke unikernel implementations. Note that unikernels, currently limited in multi-core operation capabilities, align well with Redis' predominantly single-core architecture. Figure 3 shows performance results from *X1* (`x86`) and *Pi* (`ARM`) machines. The CPU usage includes the application, the OS, the orchestration layer, and the virtualization overhead. We find that on `x86` platforms, the native unikernel variant outperforms the containerized Redis by approximately 20% and 23%, respectively, likely due to saving the cost of syscalls in unikernels [31]. In contrast, for `ARMv8`, the containerized Redis performs 20% better than the unikernel-native implementation. The containerized Redis shows 60% ↑ and 7%↑ CPU usage respectively on `x86` and `ARMv8` in comparison. Unikernel under-performing on ARM is common and caused by lack of stable support for this architecture [9].

4.1.2 ***Web Server.*** Nginx [5] is a popular production-grade web server and proxy application. We deploy 50 instances of nginx microservice on *X2* (`x86`) and perform 500 calls per millisecond from 1× stress client using round-robin load balancing. We compare the performance of nginx deployed as unikernel, as a container, and as a sandboxed container using `gVisor` [3]. The main difference between default container deployment, managed by `runc` [43], and `gVisor`, is the additional layer of isolation provided by the latter via syscall interception. Ultimately, unikernels and sandboxed containers have a comparable degree of isolation [38] – critical for services like nginx. Figure 4 shows how containerized nginx obtains the lowest latency overall, with a median of 0.25 ms per request. Unikernel and `gVisor` show higher latency, with 0.34 ms and 0.35 ms median latencies, respectively. `gVisor` shows a 22% higher tail latency at the 90*th* percentile compared to unikernel. The latency deficit for unikernels is likely due to the additional `macvtap` interface required to bridge the `qemu` to the network namespace. For `gVisor`, the overhead is due to its isolated network stack (`netstack`) inside Sentry [27]. Despite networking overhead, unikernel nginx shows lower resource consumption with 31% ↓ CPU usage compared to the container and ≈ 3.2× ↓ compared to `gVisor`.

4.1.3 ***Message Broker.*** Microservice applications rely on publish/subscribe message broker systems, such as RabbitMQ [50], for scalable communication and data exchanges. We found that popular broker frameworks are dependent on several external libraries, which necessitated us to build our own simple broker system and export it as a single binary. We package the broker as a container and unikernel using the Unikraft `elfloader` compatibility layer. Figure 5 presents the observed latency and the CPU consumption when the broker is deployed on *X1*. The CPU consumption measurements shows the CPU used by the broker service and the total CPU used by the system, including the orchestration system, the OS, etc. The latency includes the end-to-end process, from publishing a message to the broker to receiving its acknowledgment from the subscriber. We find that the unikernel broker observes 15% higher latency than the containerized version. However, its CPU usage is also lower (≈ 37%). Both virtualization targets are resource-efficient and consume < 10% total CPU. The data indicates that unikernels experience higher latency overhead, a result we again believe is linked to the overhead of network virtualization.

4.1.4 ***Hybrid Orchestration.*** We now compose the Redis, nginx, and message broker services into an IoT pipeline and evaluate the impact of hybrid virtualization on the end-to-end application performance (see fig. 7a). We deploy 100 containerized sensor nodes emulating a distributed sensor network. The sensor data is balanced between 10 nginx proxies (acting as pipeline entry points) and then pre-processed by *analytics*. The *analytics* service sends the data to the *message broker*, which can then be accessed by the *store manager* that stores the received data in the Redis database. We use `x86` machines in our infrastructure (*S1* , *S2* , *X1* , *X2*) for this experiment for comparable results across runs.

The time series in fig. 7b shows different deployment stages. At time $T = 0$, the pipeline is fully containerized. At time $T = 130$, we begin the *hot-swap* procedure where unikernel nginx and Redis are deployed alongside their container variants. The traffic gradually shifts to the unikernels thanks to the Oakestra round-robin balancing, after which the containers are undeployed. We do not *hot-swap* the broker to avoid data flow interruptions due to lost state. Figure 7b shows that though the *hot-swap* procedure temporarily increases the load by ≈ 15%, the hybrid deployment achieves ≈ 44%

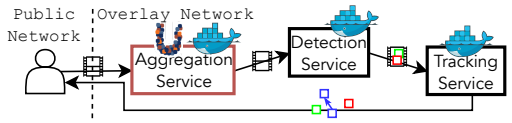


Figure 8: AR benchmark pipeline architecture.

↓ CPU usage compared to container-only. The *hot-swap* overhead is due to the parallel unikernel-container deployment, propagation of new routing rules, and the gradual shutdown of the containers. We also observe 5% higher 90*th* percentile tail latency during *hot-swap*, likely due to the unikernel transition. We find that hybrid deployments increase latency by 20% in total. The results show how hybrid orchestration reduces resource consumption and can be suitable for non-latency-critical contexts.

4.2 Hybrid Augmented Reality Pipeline

We use the AR pipeline from Comb [16], an open-source benchmark suite for edge computing. The pipeline, shown in fig. 8, is composed of the following services. *Aggregation* performs CPU-dependent video frame translation and resizing operations, *Detection* uses GPU to detect and mark objects in frames with bounding boxes, and *Tracking* uses GPU to map and track objects across frames. We re-implement the *aggregation* service with Unikraft and also package it as a container. The remaining microservices are unchanged and are deployed as containers because GPU compatibility is not yet supported by unikernels [23]. Figure 6 shows the average time required by the *aggregation* service to encode a frame (encoding time) and the internal frame processing OS system calls (processing time). The figure also shows the total time to traverse the pipeline (execution time) and the CPU utilization of the aggregation service. The containerized deployment achieves ≈ 45% lower CPU. We attribute this to the stress that latency-sensitive and high data-rate services put on the virtual network devices. The container shows ≈ 10% and ≈ 20% higher encoding and processing times, respectively, compared to unikernel for x86, and up to ≈ 28% and ≈ 70% on ARM. At the same time, the total per frame end-to-end time shows negligible < ±2% fluctuations. The reduced processing and encoding time in unikernel *aggregation* minimally impacts the end-to-end time, which is dominated by networking, inference, and decoding time. Our results highlight how latency and throughput-sensitive services suffer the overhead of unikernel network compatibility layers, a requirement for all state-of-the-art orchestration systems.

5 Discussion and Future Work

While advancements in toolchains like Kraftkit [48] are enhancing the portability of applications to the unikernel domain, not all libraries, drivers, and consequently applications are currently supported. Moreover, as we show in §4, determining the most suitable virtualization for a given application is not straightforward. Generally, we observed that within an orchestrated infrastructure, containers are a more efficient choice for network-dominant and latency-critical applications, while unikernels are better suited for CPU-intensive applications and scalability, achieving up to 44% CPU usage reduction in our cluster. It is crucial to recognize that the real-world performance of unikernels might not always align with conceptual expectations [22] and aspects such as flexibility, compatibility and security may be more relevant factors for considering optimal virtualization choice [44]. Our findings motivate for joint transparent orchestration of unikernels and containers as well as the need for a platform that abstracts its complexity. Summing up, *there is no one-size-fits-all approach for service virtualization.*

In future extensions, we envision a closer integration between Oakestra, Unikraft, and qemu to reduce the virtual network bottlenecks experienced with the unikernels. We also plan to investigate intelligent scheduling solutions for performance forecasting and a telemetry-based feedback loop for performance monitoring across virtualizations from the application to the runtime layer. Moreover, we plan to integrate cross-virtualization checkpointing to enable the data migration of stateful applications with minimal loss and downtime across different virtualization technologies.

Acknowledgments

We thank the anonymous reviewers and the shepherd for their comments and insights during the review process. This work was partly supported by the Federal Ministry of Education and Research of Germany (BMBF) project 6G-Life (16KISK002) and by the National Growth Fund through the Dutch 6G flagship project "Future Network Services".

References

[1] 2024. EVE – LF EDGE: Building an Open Source Framework for the Edge. https://www.lfedge.org/projects/eve/. Accessed: 2024-01-11.
[2] 2024. Firecracker MicroVM. http://firecracker-microvm.io/. Accessed: 2024-01-11.
[3] 2024. gVisor - The Container Security Platform. https://gvisor.dev/. Accessed: 2024-01-11.
[4] 2024. MirageOS - GitHub. https://github.com/mirage/mirage.
[5] 2024. nginx - Docker Hub. https://hub.docker.com/nginx.
[6] 2024. NVIDIA GRID vGPU User Guide. https://web.archive.org/web/20230816105820/https://docs.nvidia.com/grid/latest/pdf/grid-vgpu-user-guide.pdf. Accessed: 2024-01-11.
[7] 2024. Open Container Initiative. https://opencontainers.org. Accessed: 2024-01-15.
[8] 2024. Unikraft App for Redis. https://github.com/unikraft/app-redis. Accessed: 2024-01-11.
[9] Ashijeet Acharya, Jérémy Fanguède, Michele Paolino, and Daniel Raho. 2018. A performance benchmarking analysis of hypervisors containers and unikernels on ARMv8 and x86 CPUs. In *2018 EuCNC*. IEEE.
[10] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-time video analytics: The killer app for edge computing. *computer* 50, 10 (2017), 58–67. https://doi.org/10.1109/MC.2017.3641638
[11] Anjali, Tyler Caraza-Harter, and Michael M. Swift. 2020. Blending Containers and Virtual Machines: A Study of Firecracker and GVisor. In *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Lausanne, Switzerland) *(VEE '20)*. Association for Computing Machinery, New York, NY, USA, 101–113. https://doi.org/10.1145/3381052.3381315
[12] The Oakestra Authors. 2023. Oakestra: An Orchestration Framework for Edge Computing. https://www.oakestra.io/
[13] Giovanni Bartolomeo, Jacky Cao, Xiang Su, and Nitinder Mohan. 2023. Characterizing Distributed Mobile Augmented Reality Applications at the Edge. In *Companion of the 19th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT 2023)*. Association for Computing Machinery, New York, NY, USA, 9–18. https://doi.org/10.1145/3624354.3630584
[14] Giovanni Bartolomeo, Mehdi Yosofie, Simon Bäurle, Oliver Haluszczynski, Nitinder Mohan, and Jörg Ott. 2022. Oakestra white paper: An Orchestrator for Edge Computing. *arXiv preprint arXiv:2207.01577* (2022).
[15] Giovanni Bartolomeo, Mehdi Yosofie, Simon Bäurle, Oliver Haluszczynski, Nitinder Mohan, and Jörg Ott. 2023. Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/atc23/presentation/bartolomeo
[16] Simon Bäurle and Nitinder Mohan. 2022. ComB: A Flexible, Application-Oriented Benchmark for Edge Computing. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking* (Rennes, France)

(EdgeSys '22). Association for Computing Machinery, New York, NY, USA, 19–24. https://doi.org/10.1145/3517206.3526269

[17] Canonical. 2018. MicroK8s. https://microk8s.io. Retrieved May 24, 2022 from https://microk8s.io

[18] Shichao Chen and Mengchu Zhou. 2021. Evolving container to unikernel for edge computing and applications in process industry. *Processes* 9, 2 (2021), 351.

[19] CNCF. 2015. Kubernetes - Production-Grade Container Orchestration. Retrieved May 24, 2022 from https://kubernetes.io

[20] CNCF. 2017. KubeEdge. https://github.com/kubeedge/kubeedge. Retrieved May 24, 2022 from https://kubeedge.io/en/

[21] Vittorio Cozzolino, Aaron Yi Ding, and Jörg Ott. 2017. Fades: Fine-grained edge offloading with unikernels. In *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*. 36–41.

[22] Jonathon Robert Cross. 2021. Analysis of Unikernels for Load Balancing and Backend Service Deployment. (2021). https://jonathoncrossdissertation.s3.eu-west-2.amazonaws.com/JC-DissertationDraft.pdf

[23] Niklas Eiling, Martin Kröning, Jonathan Klimt, Philipp Fensch, Stefan Lankes, and Antonello Monti. 2023. GPU Acceleration in Unikernels Using Cricket GPU Virtualization. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis* (, Denver, CO, USA,) *(SC-W '23)*. Association for Computing Machinery, New York, NY, USA, 1588–1595. https://doi.org/10.1145/3624062.3624236

[24] Kubernetes Cluster Federation. 2023. KubeFed. https://github.com/kubernetes-sigs/kubefed.

[25] Gaulthier Gain, Cyril Soldani, Felipe Huici, and Laurent Mathy. 2022. Want More Unikernels? Inflate Them!. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 510–525. https://doi.org/10.1145/3542929.3563473

[26] Tom Goethals, Merlijn Sebrechts, Ankita Atrey, Bruno Volckaert, and Filip De Turck. 2018. Unikernels vs containers: An in-depth benchmarking study in the context of microservice applications. *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)* (2018).

[27] gVisor. [n. d.]. gVisor Networking. https://gvisor.dev/docs/user_guide/networking/. https://gvisor.dev/docs/user_guide/networking/

[28] Jin Heo, Ketan Bhardwaj, and Ada Gavrilovska. 2023. FleXR: A System Enabling Flexibly Distributed Extended Reality. In *Proceedings of the 14th Conference on ACM Multimedia Systems* (Vancouver, BC, Canada) *(MMSys '23)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3587819.3590966

[29] Vivek Jain, Shixiong Qi, and K. K. Ramakrishnan. 2021. Fast Function Instantiation with Alternate Virtualization Approaches. In *2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. 1–6. https://doi.org/10.1109/LANMAN52105.2021.9478808

[30] Gayathri Karthick, Glenford Mapp, and Jon Crowcroft. 2023. Building an Intelligent Edge Environment to Provide Essential Services for Smart Cities. In *Proceedings of the 18th Workshop on Mobility in the Evolving Internet Architecture*. 13–18.

[31] Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi Răducanu, Cristian Banu, Laurent Mathy, Răzvan Deaconescu, Costin Raiciu, and Felipe Huici. 2021. Unikraft: Fast, Specialized Unikernels the Easy Way. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 376–394. https://doi.org/10.1145/3447786.3456248

[32] Hsuan-Chi Kuo, Dan Williams, Ricardo Koller, and Sibin Mohan. 2020. A Linux in Unikernel Clothing. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 11, 15 pages. https://doi.org/10.1145/3342195.3387526

[33] Tytus Kurek. 2019. Unikernel Network Functions: A Journey Beyond the Containers. *IEEE Communications Magazine* 57, 12 (2019), 15–19. https://doi.org/10.1109/MCOM.001.1900138

[34] Hugo Lefeuvre, Gaulthier Gain, Daniel Dinca, Alexander Jung, Simon Kuenzer, Vlad-Andrei Badoiu, Razvan Deaconescu, Laurent Mathy, Costin Raiciu, Pierre Olivier, et al. 2021. Unikraft and the coming of age of unikernels. *login; The Usenix Magazine* (2021).

[35] Anil Madhavapeddy, Thomas Leonard, Magnus Skjegstad, Thomas Gazagnaire, David Sheets, Dave Scott, Richard Mortier, Amir Chaudhry, Balraj Singh, Jon Ludlam, Jon Crowcroft, and Ian Leslie. 2015. Jitsu: Just-In-Time Summoning of Unikernels. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 559–573. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/madhavapeddy

[36] Anil Madhavapeddy and David J. Scott. 2013. Unikernels: Rise of the Virtual Library Operating System: What If All the Software Layers in a Virtual Appliance Were Compiled within the Same Safe, High-Level Language Framework? *Queue* 11, 11 (dec 2013), 30–44. https://doi.org/10.1145/2557963.2566628

[37] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) than Your Container. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 218–233. https://doi.org/10.1145/3132747.3132763

[38] Ilias Mavridis and Helen Karatza. 2023. Orchestrated sandboxed containers, unikernels, and virtual machines for isolation-enhanced multitenant workloads and serverless computing in cloud. *Concurrency and Computation: Practice and Experience* 35, 11 (2023), e6365.

[39] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. 2020. Pruning Edge Research with Latency Shears *(HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 182–189. https://doi.org/10.1145/3422604.3425943

[40] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. 2018. Consolidate IoT Edge Computing with Lightweight Virtualization. *IEEE Network* 32, 1 (2018), 102–111. https://doi.org/10.1109/MNET.2018.1700175

[41] Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, and Eyal de Lara. 2017. Cloudpath: A Multi-Tier Cloud Computing Framework. In *ACM/IEEE SEC* (San Jose, California) *(SEC '17)*. Association for Computing Machinery, New York, NY, USA, Article 20, 13 pages. https://doi.org/10.1145/3132211.3134464

[42] Georgios Ntoutsos and Anastassios Nanos. 2023. URUNC: A Unikernel Container Runtime. https://osseu2023.sched.com/event/1OGgY/urunc-a-unikernel-container-runtime-georgios-ntoutsos-anastassios-nanos-nubificus-ltd. Linux Foundation Open Source Summit Europe.

[43] OpenContainers. [n. d.]. runC. https://github.com/opencontainers/runc. https://github.com/opencontainers/runc

[44] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, and Tao Li. 2019. A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms: Transparent Computing, Mobile Edge Computing, Fog Computing, and Cloudlet. *ACM Comput. Surv.* 52, 6, Article 125 (oct 2019), 36 pages. https://doi.org/10.1145/3362031

[45] W. Shi and S. Dustdar. 2016. The Promise of Edge Computing. *IEEE Computer* 49, 5 (2016), 78–81. https://doi.org/10.1109/MC.2016.145

[46] Hajime Tazaki, Akira Moroo, Yohei Kuga, and Ryo Nakamura. 2021. How to design a library OS for practical containers?. In *Proceedings of the 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Virtual, USA) *(VEE 2021)*. Association for Computing Machinery, New York, NY, USA, 15–28. https://doi.org/10.1145/3453933.3454011

[47] Unikraft. 2022. Integrations with Container Runtimes. https://unikraft.org/docs/getting-started/integrations/container-runtimes

[48] Unikraft. 2023. Kraftkit. https://github.com/unikraft/kraftkit.

[49] Vincent van Rijn and Jan S. Rellermeyer. 2021. A fresh look at the architecture and performance of contemporary isolation platforms. In *Proceedings of the 22nd International Middleware Conference* (Québec city, Canada) *(Middleware '21)*. Association for Computing Machinery, New York, NY, USA, 323–335. https://doi.org/10.1145/3464298.3493404

[50] VMware. 2023. RabbitMQ. https://www.rabbitmq.com.

[51] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 133–146. https://www.usenix.org/conference/atc18/presentation/wang-liang

[52] Dan Williams and Ricardo Koller. 2016. Unikernel Monitors: Extending Minimalism Outside of the Box. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. USENIX Association, Denver, CO. https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/williams

[53] Syed Yazdani, Naeem Ramzan, and Pierre Olivier. 2023. Enhancing Edge Computing with Unikernels in 6G Networks. In *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. 1–6. https://doi.org/10.1109/PIMRC56721.2023.10293911

[54] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. 2017. Lavea: Latency-aware video analytics on edge computing platform. In *ACM/IEEE SEC* (San Jose, California) *(SEC '17)*. ACM, New York, NY, USA, Article 15, 13 pages. https://doi.org/10.1145/3132211.3134459

[55] Wenxiao Zhang, Bo Han, and Pan Hui. 2018. Jaguar: Low Latency Mobile Augmented Reality with Flexible Tracking. In *Proceedings of the 26th ACM International Conference on Multimedia* (Seoul, Republic of Korea) *(MM '18)*. Association for Computing Machinery, New York, NY, USA, 355–363. https://doi.org/10.1145/3240508.3240561