




A Comparative Evaluation of Edge Cloud Virtualization Technologies

Polychronis Valsamas^{}, *Student Member, IEEE*, Lefteris Mamatras^{}, *Member, IEEE*, and Luis M. Contreras^{}

Abstract—Edge cloud computing is a crucial enabler for applications or network services with stringent communication requirements in the 5G networks and beyond era. Although edge compute resources contribute to a significant portion of end-to-end performance (e.g., delay), the involved performance trade-offs in relation to the utilized virtualization technologies and application functions are not yet thoroughly investigated. In this context, we study the performance dynamics of alternative edge cloud technologies, including different unikernel flavors, container builds, and implementations of exemplary web services. Our experiments do not target identifying a single best option, rather than the involved performance trade-offs with respect to both service and infrastructure viewpoints as well as realistic edge cloud tasks, including on service operation, cloud resource or service elasticity, dynamic resource allocation and removal. We provide insights gained from a relevant extensive comparative evaluation based on a novel edge cloud experimentation infrastructure. Our experimental results inspired a conceptual edge cloud orchestration platform and its basic design guidelines, i.e., mixing and matching heterogeneous edge cloud virtual entities with particular operational or performance requirements.

Index Terms—5G Networks and Beyond, Containers, Unikernels, Comparative Evaluation, Edge Cloud Orchestration

I. INTRODUCTION

5G networks and beyond (5GB) [1] are being characterized by significant network performance and capacity advantages, especially at the radio level. They devised a promising agenda targeting new applications that enable a radical transformation of vertical sectors, including manufacture, media & entertainment, health, energy and automotive industry. Such services should adhere to stringent requirements, e.g., ultra-low delay or high throughput, scalable operation, and increased adaptability to dynamic contexts, in terms of service needs or resource availability. In other words, there is a need for systemic adaptations of 5GB ecosystems towards these goals. For example, virtual resources contribute significantly to end-to-end (E2E) service performance. Indicatively, for a 50ms E2E delay documented in paper [2] on 5G networks, radio and transport aspects caused the 17% of delay, while cloud dimension the 83%.

Along these lines, edge cloud computing is an important enabling technology for 5GB ecosystems, bringing computation close to end-users to improve service performance, reliability and data privacy of users. Thus, how to properly exploit the available computing substrate is a major concern

for network operators with respect to the overall network and service management. Several aspects can be considered to take informed decisions for the usage of this infrastructure. Clearly, the performance improvement that can be obtained from using distinct compute facilities, the availability of resources as the services become deployed, the need for elasticity to accommodate a variety of application requirements (as anticipated in 5G), as well as the particular virtualization solutions that can be put in place to account for all of this.

For instance, large-scale service deployments usually serve a vast amount of users spread throughout the globe, which require the involvement of edge cloud resources in many different places. However, it is challenging to deploy resources near every user, consequently, there is a need for optimized cloud facilities not only towards particular performance requirements, but also mitigating a potential limited resource availability. Such infrastructures should be able to mobilize any available deployment, even with alternative server configurations, as well as network or virtualization technologies.

Given the fact that user demands or application requirements may be dynamic, edge clouds should also support a quick deployment or removal of virtual resources, implementing horizontal and vertical elasticity processes, i.e., adapting the service deployment and cloud resources to these requirements [3], respectively. For example, legacy cloud deployments may be using inefficient virtualization technologies, including traditional virtual machines (VMs), that face slow times for deployment, downloading or scaling up of virtual resources.

There is an on-going effort towards adopting lightweight virtualization approaches for edge clouds, such as containers [4] and unikernels [5]. For example, European 5G-PPP initiative investigates alternative container and unikernel approaches to be used for edge computing [6]. In our experience, different container builds or unikernel flavors exhibit diverse performance capabilities. For example, containers can achieve a robust operation, while unikernels have rapid manipulation capabilities, e.g., they can boot up just in ms, even with a TCP SYN or DNS lookup request packet [7].

We argue that there is no single best virtualization solution for edge clouds, but the choice depends on the particular requirements, suggesting that the selection of the most appropriate approach should become an important overall management and operation task. Furthermore, a number of papers (e.g., [8], [9]) perform comparative evaluations of different virtualization technologies, but, in our understanding, none considers all basic tasks of edge cloud deployments, including elasticity of service nodes and physical resources, as well as service operation.

Here, we conduct a systematic experimental evaluation of

Polychronis Valsamas and Lefteris Mamatras are with the Department of Applied Informatics, University of Macedonia, 156 Egnatia Str., 54636, Thessaloniki, Greece (e-mails: xvalsama@uom.edu.gr, emamatras@uom.edu.gr).

Luis M. Contreras is with Telefónica I+D / CTIO, Madrid, Spain (e-mail: luismiguel.contrerasmurillo@telefonica.com).

alternative container and unikernel builds of exemplary web services towards improving performance and adaptability of edge clouds. We bring a number of novelties, including on:

- introducing a *novel edge cloud experimentation environment*, supporting load prediction and balancing, horizontal and vertical elasticity, as well as common abstractions and APIs over heterogeneous virtual resources;
- providing an *extensive experimentation analysis* of different lightweight virtualization options for edge clouds, considering all basic edge cloud processes (i.e., resource allocation, removal, service operation, horizontal and vertical elasticity actions).
- presenting *basic design guidelines of edge cloud orchestration systems*, backed by our results and highlighted through a relevant conceptual facility that supports container and unikernel-based virtualization technologies as well as alternative service node implementations, while exploiting their diverse performance characteristics.

Our experimentation exercise was challenging, since it required the implementation of a complete edge cloud orchestration solution that supports heterogeneous virtualization choices, including the production-ready containers and alternative experimental implementations of unikernel flavors, often with bugs and instabilities. We required coding in many environments, i.e., C, Python, NodeJS, OCaml, the implementation of a bespoke DNS server, as well as a number of non-trivial OS, network, hypervisor and test-bed configurations.

The remainder of the paper is organized as follows. Section II provides an overview of the related investigations. Section III details our experimentation environment. Section IV elaborates on our methodological approach and its relevant assumptions. Section V provides our experimentation analysis and produced insights, focusing on the deployment, removal, operation and elasticity of edge cloud resources and services. Section VI provides basic design guidelines for a novel edge cloud orchestration system that benefits from our findings. Finally, Section VII concludes the paper.

II. RELATED WORKS

Next-generation services call for network and cloud paradigms that enable ultra-low latency or high-throughput communication and bring elasticity in the service operation. For example, a number of approaches particularly focus on speeding-up packet processing, including the novel in-kernel proposals of extended Berkeley Packet Filter (eBPF) and Xpress Data Path (XDP) [18]. Furthermore, the edge cloud infrastructure StarlingX [19] targets at achieving ultra-low latency of network services through operating on top of real-time linux, employing Time-Sensitive Networking [20] capabilities.

Regarding the cloud viewpoint, 5G networks are gradually employing virtualization [21] and edge cloud technologies [22], as well as the microservice paradigm [23]. However, edge clouds may be associated with limited resource availability or dynamic service demands or network conditions, highlighting the need for flexible, lightweight virtualization technologies [6] at the edge, being efficiently orchestrated.

The main candidates for lightweight virtualization in edge

clouds are containers and unikernels. Containers (e.g., Docker [24] or LXC [25]) are standardized units implementing application packaging with all of its dependencies, providing robust performance and adaptability to dynamic application requirements. Unikernels [5] (e.g., MirageOS [26], ClickOS [27], RumpKernel [28], or OSv [29]) are single-purpose appliances specialized at compile-time into standalone kernels, characterized by very low resource usage and rapid deployment capabilities, even at the range of ms [7].

Several studies conduct performance comparisons of alternative lightweight virtualization options, being suitable for edge cloud deployments. An overview of such works is shown in Table I, highlighting the aspects being evaluated (i.e., on service operation and elasticity / fault-tolerance behavior), the considered services, as well as the numbers and types of contrasted virtualization options.

As enlisted in the Table, a number of considered works (i.e., [8] - [13]) assess the performance of alternative lightweight virtualization technologies and focus on investigating service operation aspects only. For example, the comparative analysis [10] focuses mainly on server resource-efficiency aspects (i.e., memory footprint and network latency) and contrasts alternative unikernel flavors with containers hosting an Nginx web-server or a Redis database. Most unikernel flavors perform at least equally or better than containers, especially in cases that require the transfer of unikernel or container images.

Other proposals consider both server resource-efficiency and service performance. Paper [8] compares KVM VMs, RumpKernel unikernels and docker containers hosting an Apache web-server or a Redis database with different numbers and sizes of requests. According to their results, containers achieve the best performance, in terms of communication delay and server resource utilization. Proposal [11] evaluates the performance of a unikernel-based firewall service against corresponding container and Linux-based solutions, concluding that the first option achieves a higher number of TCP requests served per second and a lower network latency. Similarly, paper [12] compares the network performance (i.e., requests served per second and latency) of unikernels against linux-based solutions offering both DNS and web-based services.

Furthermore, paper [13] evaluates the performance of unikernels versus containers for the same REST service, implemented in Java, Go, and Python. This study measures memory consumption and execution / response times and concludes that unikernels perform at least equally or outmatch the corresponding containers, however, the former consumes significantly more memory compared to the latter. Along the same lines, the authors of [9] compare the HTTP and database access performance of OSv, RumpKernel, MirageOS, and IncludeOS unikernels as well as docker containers, reporting a higher request rate in the case of containers, but a lower latency when employing unikernels.

A number of proposals consider aspects of elasticity or fault-tolerance processes. In [14], the authors compare KVM-based VMs, Docker containers and OSv-based unikernels in an OpenStack cloud platform, documenting that OSv outperforms the other virtualization technologies in terms of service provisioning time. A Vehicle Ad-Hoc Network realizing a

TABLE I: Related works comparing alternative virtualization technologies that are suitable for edge cloud environments

	Edge Cloud Aspects						Service Type	Number of considered container technologies	Number of considered unikernel technologies	Alternative flavors of the same application
	Service Operation		Elasticity or Fault-tolerance							
	Server Resource-efficiency	Service Performance	Resource Allocation Time	Resource Removal Time	Server Resource-efficiency Impact	Service performance Impact				
[10]	✓						Web-service & Database	2	2	
[8]	✓	✓					Web-service & Database	1	1	
[11]	✓	✓					Firewall	1	1	
[12]	✓	✓					DNS & Web-service		2	
[13]	✓	✓					HPC	1	1	3
[9]	✓	✓					Web-service & Database	1	4	2
[14]			✓				Virtual Entity Instantiation	1	1	
[15]			✓				Network Memory Server	2	1	
[16]	✓	✓	✓				Firewall	1	1	
[17]	✓	✓	✓				Web-service	2	1	
Ours	✓	✓	✓	✓	✓	✓	Web-service	1	3	2

service migration scenario and utilizing KVM-based VMs, both LXN and docker containers, as well as OSv-based unikernels is considered in [15]. The same work assesses the alternative virtualization options in terms of allocation time of a simple Network Memory Server and demonstrate the lower service allocation time of the unikernel option.

In our understanding, two of the related works consider both service operation and elasticity, however taking into account only the resource allocation aspect of the latter. In [16], the authors conduct a performance comparison between unikernel and container based implementations of a firewall, in the context of a fault-tolerance scenario, reporting that containers exhibit the best network performance (i.e., in terms of latency and throughput) and service instantiation time. Paper [17] compares a traditional VM with alternative container technologies and a unikernel-based implementation, all hosting a web-service. The last option achieves the lower boot-up times and higher network throughput, while the container option the best CPU and memory consumption efficiency.

As we see in Table I, all the aforementioned studies consider one or two virtualized network services (or application functions) with the web-service being the most commonly used, since it may represent a number of REST-based services. We also observe that almost all of them investigate containers, mostly docker, while three of them (i.e., [10], [15] and [17]) consider alternative container virtualization technologies. Furthermore, papers [9], [12] and [10] assess different unikernel flavors, while works [9] and [13] consider multiple implementations of the same application.

In summary, the relevant papers provide comparative evaluations of alternative lightweight virtualization technologies with respect to service operation and/or assurance aspects. However, the current paper is the only one that considers all of these aspects, including virtualized service deployment and

removal times, as well as the impact of elasticity processes on server resource-efficiency and service performance. In contrast to the related works, our analysis (i) clearly targets the specific context of edge clouds; (ii) focuses on all basic relevant service and cloud operations; (iii) considers containers, multiple unikernel flavors and implementations of the same service; and (iv) targets to identify the performance trade-offs of alternative lightweight virtualization options, so they can be appropriately tuned, even by mixing multiple technologies.

III. EDGE CLOUD EXPERIMENTATION ENVIRONMENT

Here, we detail our experimentation infrastructure, including its basic components and interactions.

Our edge cloud experimentation environment is investigating the proposed edge cloud paradigm and its core design requirements, while demonstrating the following novelties: (i) implements all basic edge cloud operations, including the deployment, removal and operation of virtualized web-based services; (ii) realizes web load prediction and balancing, as well as both horizontal and vertical elasticity; (iii) employs heterogeneous lightweight virtualization technologies as well as different implementations of particular applications to realize adaptability of real deployments in various circumstances, e.g., rapid changes in users requesting content or resource availability; and (iv) is extendable to support new services, virtualization technologies, orchestration workflows and corresponding mechanisms.

Our experimentation facility (i.e., Fig. 1) comprises of three node clusters: the centralized (i.e., purple colored), the edge (i.e., blue colored), and the client nodes (i.e., green colored), all residing at our SWN test-bed [30]. We detail the three parts of our infrastructure and their basic components below, as well as their basic interactions.

The centralized node accommodates the *Service Orches-*

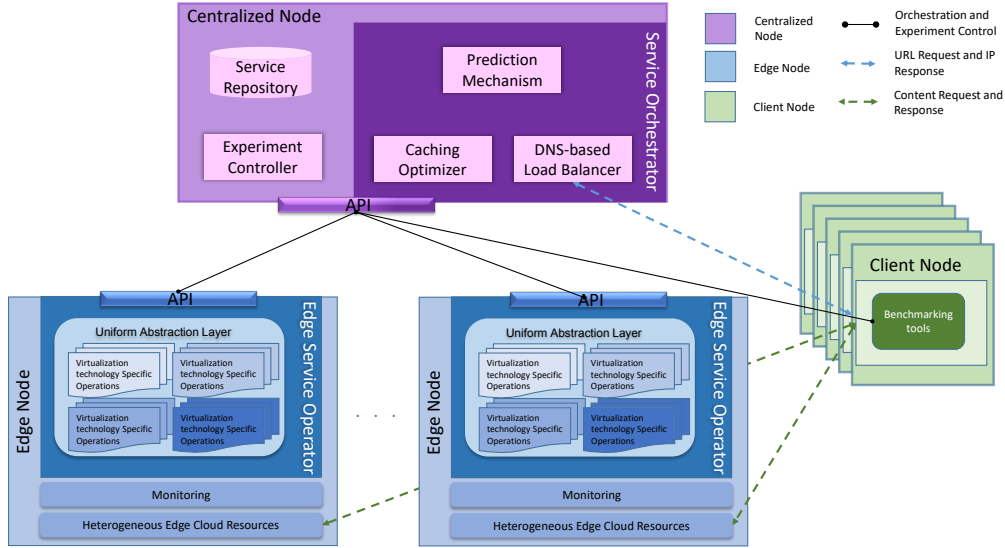


Fig. 1: Experimentation environment

trator, the *Experiment Controller* and the *Service Repository*. The *Service Orchestrator* is implemented in Node-RED [31], a modular programmable environment utilizing a browser-based flow editor. As shown in the top of Fig. 1 it comprises of three standalone components: (i) the *DNS-based Load Balancer*; (ii) the *Prediction Mechanism*; and (iii) the *Caching Optimizer*. A brief presentation of these components follows:

- The *DNS-based Load Balancer* realizes load balancing over heterogeneous virtual resources, i.e., assigns client requests to particular servers, in a round-robin fashion. It supports heterogeneous virtual resources through different IP subnets. For simplicity, the URL indicates the content requested and the type of virtual resource. It maintains a list of active nodes serving content in cooperation with the *Caching Optimizer*, i.e., the latter provides notifications on all additions or removals of virtual resources. *DNS-based Load Balancer* is also tracking the content requests over fixed time intervals, i.e., 30 sec, in our case. Such information is the input of the *Prediction Mechanism*.
- The *Prediction Mechanism* utilizes the client requests' status from the previous component to predict the forthcoming load, based on historical information it maintains. Currently, we support two alternative load prediction mechanisms, the *Exponential Moving Average (EMA)* and the *Seasonal Autoregressive Integrated Moving Average (SARIMA)* [32]. *EMA* considers the recent measurements as more significant and its formula is: $EMA_t = \frac{2}{n+1} * R_t + (\frac{n-1}{n+1}) * EMA_{t-1}$ (R_t expresses the current requests' value). *SARIMA* extends *ARIMA* to consider seasonal trends, defined as $SARIMA(p,d,q)(P,D,Q)m$, where the parameters in the first (i.e., p, d, q) and the second set of brackets (i.e., P, D, Q) indicate the trend and seasonal parameters (i.e., autoregression, difference, and moving average orders), respectively. The mechanisms are implemented as NodeRED modules, so it is straightforward to introduce new models, however this paper is not focusing

on prediction aspects.

- The *Caching Optimizer* is responsible for controlling the virtual resources, including determining the quantity and location of resources to be deployed or removed, depending on the input of *Prediction Mechanism*. In our experimental analysis, we assume that each VM or container can serve up to a fixed number of requests. Consequently, the number of resources to be deployed or removed is calculated from the *Caching Optimizer* based on the estimation of the upcoming content requests and the total existing capability of the edge nodes to handle the web load, i.e., by subtracting the latter from the former and then dividing the result by the above fixed number. In case the result is negative or positive, a scale up or down event is triggered, respectively. The *Caching Optimizer* decides to deploy new service entities to the servers with the lowest number or remove existing ones from the nodes with the maximum number of entities, i.e., due to the homogeneous hardware of our test-bed servers. However, the *Caching Optimizer* receives not only the status of virtual entities but also recent resource allocation monitoring information from all nodes, which could be the basis for more sophisticated placement mechanisms. In case of a new deployment, it first confirms that the new virtual resource is up and running (i.e., through the *Edge Service Operator*) and then communicates the new IP address to the *DNS-based Load Balancer*. Whenever it removes an existing resource, it first notifies the latter, so no new users request content, waits for existing communication to complete, and then removes the particular virtual entity.

The centralized node also accommodates the *Experiment Controller* and the *Service Repository*. The former component utilizes custom scripts specifying the configuration of experiments (i.e., initiates cloud resources and client requests) and collecting the results from both client and edge nodes through the API interface. The latter stores, locates and communicates

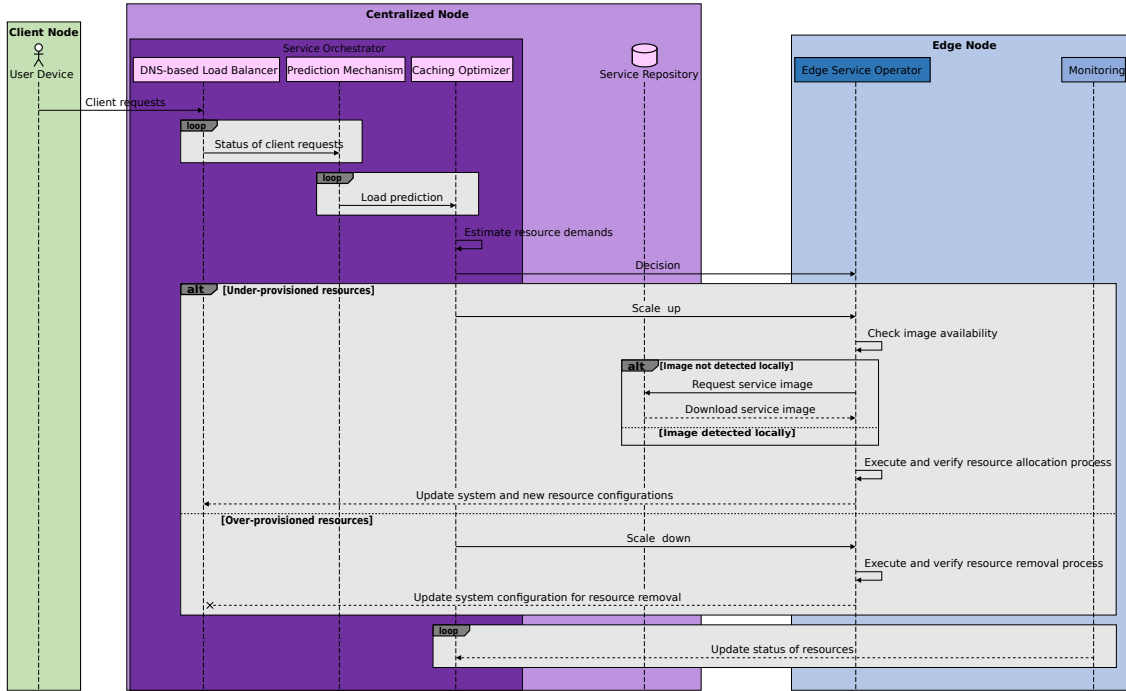


Fig. 2: Elasticity workflow

both unikernel and container images based on a common *Service Repository* API for heterogeneous virtual resources. It is built on top of a private Docker repository and a custom script handling unikernel images. The repository is utilized whenever a horizontal elasticity event is triggered, i.e., virtual resources are being deployed in new edge nodes, otherwise images are already hosted by the latter.

The second part of our facility contains the edge nodes' cluster. Each edge node is equipped with an *Edge Service Operator* and a *Monitoring* component. The former is a standalone software entity being responsible for the manipulation and configuration of edge cloud resources. It uses a uniform abstraction layer that hides the heterogeneity of virtualization resources. For example, the *Service Orchestrator* communicates general requests to deploy virtual resources to the *Edge Service Operator*, which in turn performs the following tasks: (i) locates corresponding virtualization-technology-specific APIs; (ii) identifies the location, i.e., being local or remote, and the details of required images; (iii) allocates the particular resources and assigns IPs to them; (iv) confirms the completion of deployment through a frequent polling process by requesting a tiny-sized content (i.e., every 0.1 sec); and (v) notifies the *DNS-based Load Balancer* upon the completion of the operational task through the API. Finally, the *Monitoring* component collects real-time data information about the physical and virtual resources in terms of resource availability and the status of edge cloud virtual resources.

The last part of our experimentation environment consists of the client nodes hosting our *benchmarking tools*, a custom multi-threaded workload generator tool being responsible for emulating the clients' behavior as well as assessing their performance with different metrics.

In Fig. 1 we also highlight the basic connectivity among the components of the experimentation facility. The centralized node communicates with both edge and client nodes through the API interface for orchestration, monitoring information and experimentation control processes, e.g., the implementation of elasticity events or changes in the configuration of the experiment.

For example, message exchange sequence diagram of Fig. 2 illustrates the interactions between the platform components for the realization of an elasticity process. In this workflow, *DNS-based Load Balancer* keeps track and notifies the *Prediction Mechanism* for the status of content requests, which in turn may request a scale up or down event from the *Caching Optimizer*, i.e., through an *Edge Service Operator*. The latter component downloads the required service image from the *Service Repository*, in the case it is not available locally, and then boots up and verifies a corresponding resource allocation or removal process. The elasticity event completes with the necessary configurations. Lastly, the *Monitoring* component informs periodically the *Caching Optimizer* for the status of cloud resources.

In the next section, we proceed describing our methodological approach.

IV. METHODOLOGY AND ASSUMPTIONS

Here, we detail our methodological approach and provide the relevant assumptions, including on the considered service and virtualization technologies, networking aspects, as well as on the metrics used and the statistical evaluation of our results.

In our investigation, we consider a service that resembles a content delivery platform (e.g., distributing videos, music or other content) or a microservice / network service that hosts a

TABLE II: Association of considered metrics with related works' categorization criteria

Metrics	Edge Cloud Aspects					
	Service Operation		Elasticity or Fault-tolerance			
	Server Resource-efficiency	Service Performance	Resource Allocation	Resource Removal	Server Impact	Service Impact
Resource Allocation Time			✓			
Resource Removal Time				✓		
CPU Service and Infrastructure Utilization					✓	
CPU Peak	✓	✓				
Response Time		✓				✓
Download Time		✓				✓
Total Delivery Time		✓				✓
Network Throughput		✓				

local database and transmits messages via a REST interface. For simplicity, we deploy web servers able to transmit content or messages of different sizes. We also assume that content is embedded in the VMs or containers, so content size impacts relevant image sizes. Web servers are typically used from microservices and are also supported by all virtualization approaches, e.g., even from unikernel flavors at their early implementation stages, consequently serving as a good basis for our comparisons.

Currently, the edge nodes support three common unikernel options, i.e., ClickOS, RumpKernel, and MirageOS. We integrated Nginx web servers to the first two, due to its relative simplicity, high-performance and minimalistic size. We use a simple OCaml-based web server for MirageOS, since it does not support Nginx. Furthermore, we built two lightweight Docker container images, the one hosting an Nginx web server and the other the Flask Python web framework, i.e., to be able to assess also the impact of web server type, marked as *C_Nginx* and *C_Flask*, respectively. Table III illustrates the VMs and containers' image sizes we utilized in our experiments, concerning the particular content cache sizes. In our experiments, all unikernel and container technologies are resource-limited to one vCPU and 256MB of RAM. We also set the maximum number of content requests served by all virtualization technologies as 20.

We are currently assuming content requests equally spread among client nodes based on particular patterns and organized in periodic batches. Due to space constraints, we left a more thorough study on the impact of different user patterns on the performance of the proposed paradigm as a future work, since the current assumption suffices to highlight the particular novelties identified in the paper.

We implement E2E communication between client and edge nodes, which spans over both physical and virtual networks. Each client retrieves content after looking up the particular URL through the *DNS-based Load Balancer*, which connects the client to an appropriate virtual server node. The E2E path is configured through static routes generated by a bespoke script and the virtual network is implemented through both XEN and Docker bridging. For better performance, we disabled the Spanning Tree Protocol (STP) in the virtual network bridges. We also configure all network interfaces with the traffic control (tc) tool to align the network configuration with the scale of our experiments.

Furthermore, we extract measurements from both client

TABLE III: Image sizes of different virtualized services

Content	C_Nginx	C_Flask	ClickOS	RumpKernel	MirageOS
1MB	22.5MB	85.2MB	6.2MB	34MB	13.4MB
5MB	26.5MB	89.2MB	9.4MB	37.8MB	16.7MB
10MB	31.5MB	94.2MB	14.6MB	42.6MB	20.6MB
20MB	41.5MB	104MB	24.5MB	52.1MB	26MB

and edge nodes on service fulfillment, service assurance and communication performance, as summarized below:

- *service fulfillment* with the following metrics: (i) *Resource Allocation Time* is the total duration, in seconds, required for the deployment of a VM or a container, i.e., the amount of time from the *Service Orchestrator* triggering its deployment until it is ready to serve content; and (ii) *Resource Removal Time* expressing the time in seconds to remove a VM or container.
- the *service assurance* metrics applied here are: (i) *CPU Service and Infrastructure Utilization* measuring the CPU utilization of VM or container providing the service and of the edge node, respectively; and (ii) *CPU Peak* quantifying the highest point in CPU utilization in the life-span of the corresponding task, i.e., focusing on worst-case CPU utilization.
- *communication performance* with the metrics: (i) *Response Time* expressing the time, in seconds, passed from the client request to the receipt of the first byte of the response; (ii) *Download Time* representing the total duration, in seconds, required for the transfer of content to the client; (iii) *Total Delivery Time* reflecting the aggregation of *Response* with *Download Time*; and (iv) *Network Throughput* which is the ratio of transmitted data to download time, in MB/s;

Service fulfillment and assurance performance metrics are tightly associated with the service lifecycle that ensures utilizing the necessary resources and maintaining service quality, i.e., also reflected in the communication performance, focusing on the content delivery aspect, in our case. We use a different categorization for the metrics from the considered edge cloud operations, since some metrics are used to assess more than one operation. For clarity, in Table II we associate the considered metrics with the categorization criteria of our related works investigation, as presented in Table I.

Finally, we evaluate our results' statistical accuracy by executing each experiment for an indicated number of times that produces low standard deviations between the replicated runs. All provided figures represent the average values over these runs as histogram plots. For simplicity, we provide the

upper bounds of the associated standard deviations, whilst in particular cases, i.e., further supporting the statistical accuracy of our findings, we illustrate our results as box plots that provide supplementary statistical information, such as mean, median, outliers, lower and upper quartiles. Next section provides our experimental analysis that is based on the presented experimentation platform and the discussed methodology.

V. EXPERIMENTATION RESULTS

In this section, we validate the main concepts behind the proposed edge cloud paradigm, including the utilization of lightweight and heterogeneous virtual resources, as well as identify strategies for efficient resource allocation and service performance. Our analysis is organized into three scenarios covering essential edge cloud processes: (i) the *Deployment and Demobilization* scenario assessing the deployment and removal of diverse virtualized service entities; (ii) the *Service Operation* scenario quantifying the content delivery performance of alternative unikernel flavors, containers and web server technologies; and (iii) the *Elasticity* scenario considering a complete elasticity workflow that involves the prediction of client demands and a proactive strategy deploying content at the edge cloud. We measure *service assurance* in all scenarios, *service fulfillment* in the first one and *communication performance* in the other two. Our results and produced insights follow.

A. Scenario 1: Deployment and Demobilization

In the first scenario, we assess the individual performance characteristics of diverse virtualization and web server technologies during their deployment or removal, while ranging content size from 1 to 20MB, i.e., impacting VM or container image sizes. We deploy and remove one virtual resource at a time and wait for 3 minutes between different deployments, i.e., to cool off the CPU. We validated the statistical accuracy of our results by replicating each experiment 30 times.

The scenario includes the following two groups of results on (i) *vertical involvement of cloud resources*, assuming the existence of unikernel or container images for the service provisioning, at allocated physical servers; and (ii) *horizontal involvement of cloud resources*, representing the outcome of a horizontal elasticity process allocating new physical resources that should also download service images from the *Service Repository*, residing at the *centralized node*, i.e., assuming that new resources are available instantly. The resource allocation and removal processes concern the booting up or removal of both corresponding virtual entities and web servers.

In the case of vertical involvement of cloud resources (i.e., Fig. 3), ClickOS outperforms other approaches in terms of *Resource Allocation Time*, succeeded by C_Nginx (i.e., Fig. 3(a)). For example, ClickOS, in the best case (i.e., of 1MB), can be deployed 4.6 times faster than MirageOS, while in the worst case (i.e., of 20MB) achieves around 15% lower *Resource Allocation Time* from C_Nginx. The improvement ratio of ClickOS is slightly canceled as the content size increases, besides the comparison with RumpKernel. In all circumstances, MirageOS faces the worst performance,

however, associated with the most efficient CPU utilization (i.e., Fig. 3(b)). We also notice that *Resource Allocation Time* of MirageOS is not affected by content size. The lowest *CPU Peak* value of MirageOS can be justified from its prolonged boot time. We also see in Fig. 3(b) a *CPU Peak* difference between unikernels and containers ranging between 1% and 8%, in favor of unikernels. MirageOS and ClickOS have an almost steady increase of *CPU Peak* per content size increment, while in the other options appears a more fluctuated behavior. The standard deviation between the runs regarding the *Resource Allocation Time* and *CPU Peak* does not exceed the value of 0.19 and 0.36, respectively in all cases.

In Fig. 3(c), we depict the *Resource Removal Time* with an equivalent experiment to the above. We see that such metric is not influenced by the content size, in all virtualization and web server options. Furthermore, all unikernel flavors can be removed at almost zero time, highlighting their significant performance advantages in scaling down processes. Fig. 3(d) illustrates as a box plot the associated *CPU Peak* values with the resource removal process. We observe a few outlier values, mostly in the case of ClickOS, and mean / median values that are roughly similar. The results appear symmetrically distributed in all virtualization options, where MirageOS prevails with lower *CPU Peak* values for 5 to 20MB content sizes. Complementary, we observe that unikernels are approximately characterized by 1% to 4% lower mean *CPU Peak* values, compared to container options, mainly caused by their more lightweight processes.

Regarding the horizontal involvement of cloud resources, we observe that ClickOS achieves overall the best performance for all content sizes, in terms of *Resource Allocation Time* (Fig. 4(a)). For example, ClickOS is characterized by a 3.7 times lower boot-up time compared to C_Flask, in the best case (i.e., with 1MB content), while RumpKernel attains close results (i.e., the 91%) with a 10MB content size. We also notice a symmetrical distribution of results with ClickOS being associated with a higher variability (i.e., with 1 and 5 MB content), i.e., reflected in a inter-quartile range around 2. We also notice a steady improvement ratio reduction with the gradual increase of content size, i.e., when comparing ClickOS with all other solutions besides Rumpkernel. The standard deviations of results on *Resource Allocation Time* do not exceed value 0.92.

Furthermore, as shown in Fig. 4(b), MirageOS contributes to a lower *CPU Peak*, in each case. This can be justified again due to the prolonged booting up time of MirageOS. Additionally, the high *CPU Peak* of C_Flask may be attributed to its large image size. We also observe that unikernels are more resource-efficient than containers in terms of CPU utilization, as reflected in their at least 20% lower *CPU Peak* values. This difference is much higher than the case of vertical involvement of resources and is attributed to the CPU consumption required for the image downloading process. We also notice an almost steady increase in *CPU Peak* with content size, for all virtualization cases. The standard deviation of *CPU Peak* values ranges from 0.6 to 1.5.

Here, we see that VM or container image download time impacts these results, gradually canceling the advantages of

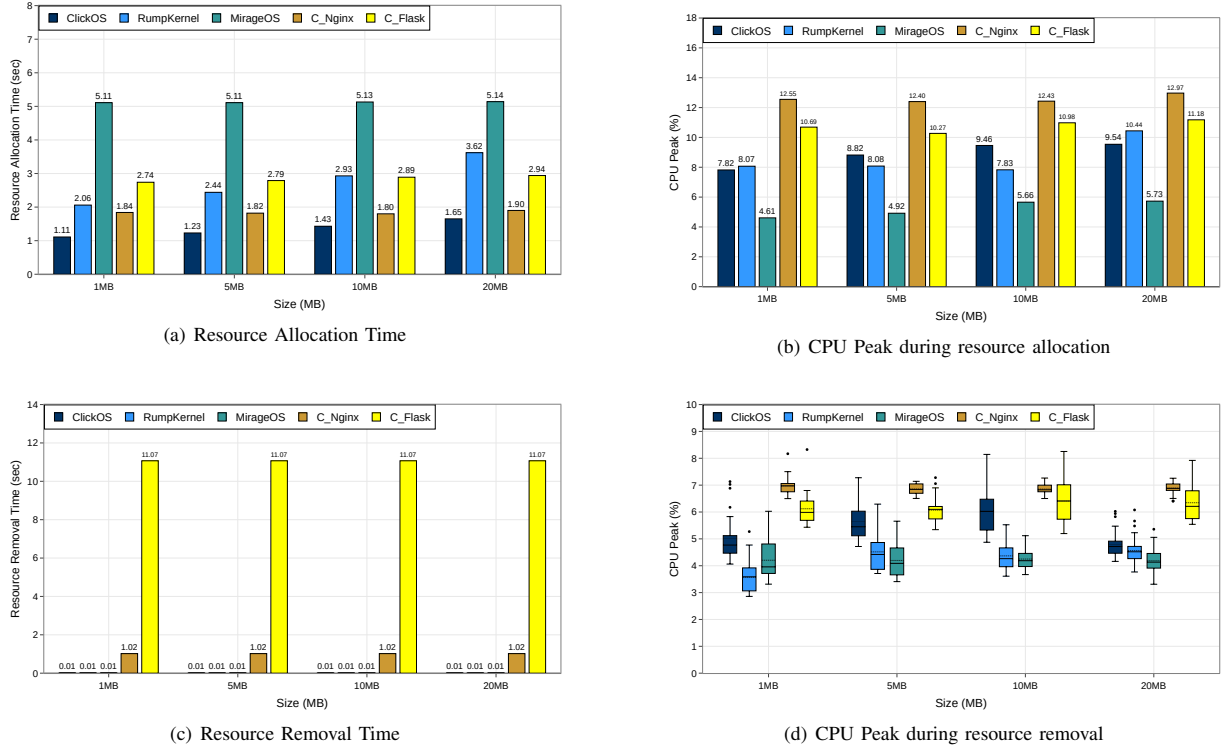


Fig. 3: Service fulfillment of alternative virtualized services - vertical involvement of cloud resources

tiny VM sizes (or virtualization solutions with rapid boot up times) as content size increases, i.e., with content that is either embedded or downloaded together with the images.

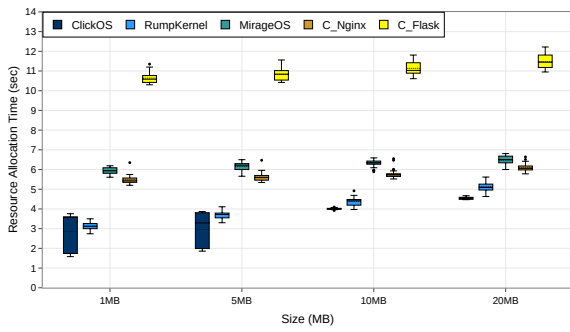
As we show in Fig. 4, the *Resource Removal Time* has an almost identical behavior as in Fig. 3(c), highlighting once more the significant performance advantages of unikernels in scaling down processes. Furthermore, RumpKernel is characterized by the lowest *CPU Peak* value (i.e., Fig. 4(d)) for removing resources with lower-sized content, while MirageOS slightly outperforms RumpKernel for larger-sized content. We also note that all unikernel options have equivalent *CPU Peak* values, which are independent of the content size, while *CPU Peak* in containers marginally oscillates. Complementary, we observe that unikernels achieve up to 6% lower *CPU Peak* values compared to containers. This could be attributed to the different shutting down processes between Docker and XEN, rather than to the content size. In overall, the standard deviations of *Resource Removal Time* and *CPU Peak* do not exceed the values of 0.01 and 1.1, respectively.

Here, we summarize our findings from the first scenario: (i) both unikernel and container-based options can be quickly manipulated in the context of a vertical involvement of cloud resources, with ClickOS achieving the best resource allocation time (i.e., at least 15% lower in contrast to C_Nginx); (ii) unikernel options consume around 1% to 8% less CPU resources during their deployment compared to containers, with MirageOS consuming at least 3% and 6% lower CPU resources in contrast to other unikernel flavors and containers, respectively; (iii) for horizontal elasticity allocation processes,

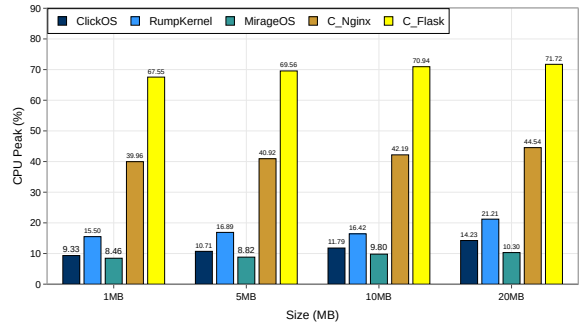
the required image downloading brings further advantages to unikernels with respect to containers, in terms of *CPU Peak* and *Resource Allocation Time*, due to their tiny sizes, e.g., MirageOS achieves an at least 31% lower *CPU Peak* value and ClickOS boots-up at least 1.35 times faster from C_Nginx, respectively, however these are gradually canceled with the content size; (iv) in both vertical and horizontal involvement of cloud resources all unikernel options can be instantly removed, i.e., in 0.01 sec, independently of the content size, while achieve up to 6% lower *CPU Peak* values compared to containers.

B. Scenario 2: Service Operation

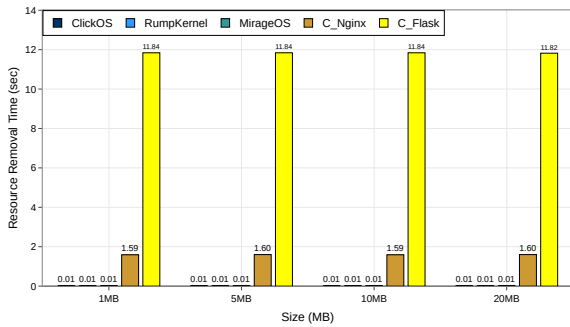
In the second scenario, we evaluate the operation of the assumed content delivery service in terms of resource efficiency and performance with all considered virtualization and web server options, despite MirageOS. We omitted the latter for two reasons: it did not perform well in the previous scenario and it faced stability issues. We also ranged the content sizes from 1 to 20MB. 10 replications of the runs sufficed for statistically accurate results. In the case of *Response* and *Download Times*, we calculate their standard deviation among different batches rather than of individual client requests, since they exhibit relatively high fluctuations due to the network. All metrics have an equivalent behavior for 5MB, 10MB and 20MB content. For this reason and due to space constraints, we omitted the 5MB and 10MB results for *Network Throughput* and *CPU Peak*.



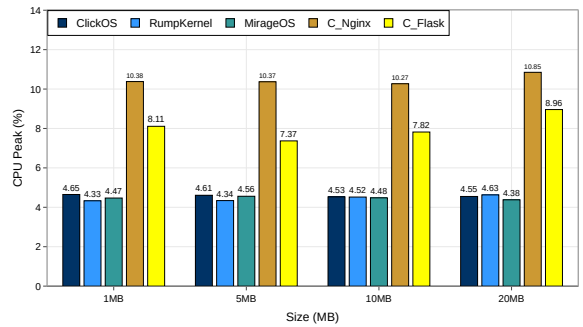
(a) Resource Allocation Time



(b) CPU Peak during resource allocation

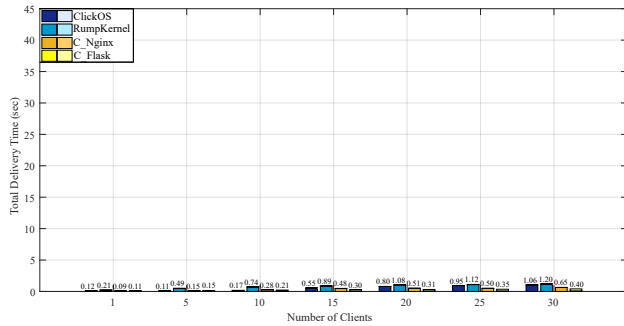


(c) Resource Removal Time

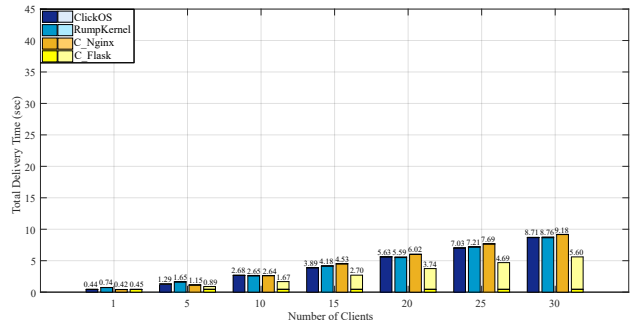


(d) CPU Peak during resource removal

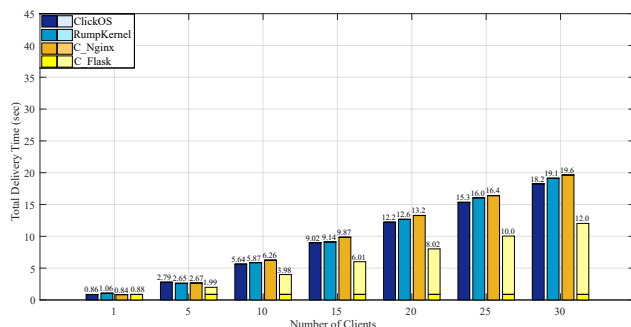
Fig. 4: Service fulfillment of alternative virtualized services - horizontal involvement of cloud resources



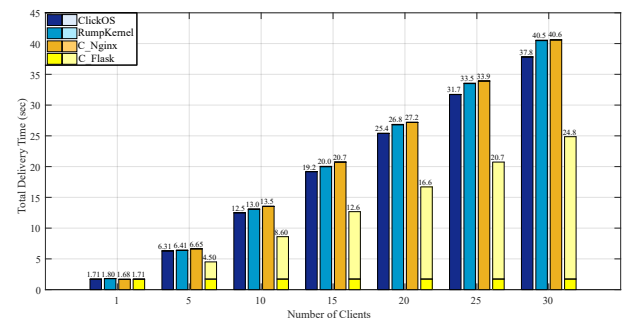
(a) Clients' Total Delivery Time with content size 1MB



(b) Clients' Total Delivery Time with content size 5MB



(c) Clients' Total Delivery Time with content size 10MB



(d) Clients' Total Delivery Time with content size 20MB

Fig. 5: Communication performance - Total Delivery Time as Response (light colored) plus Download Time (dark colored)

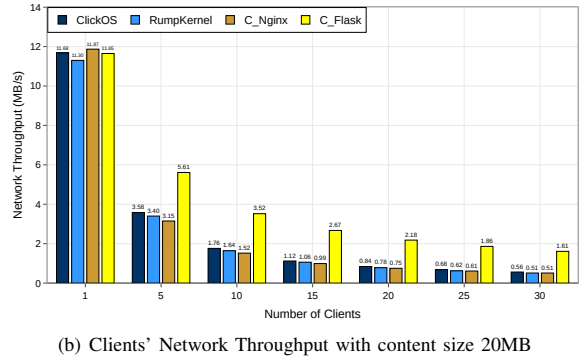
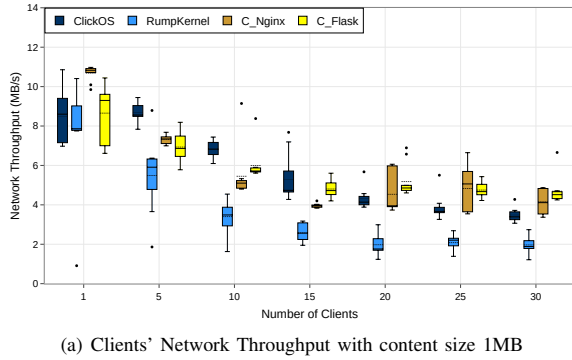


Fig. 6: Communication performance - Throughput

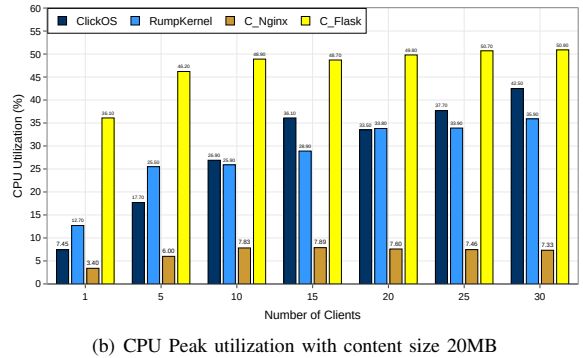
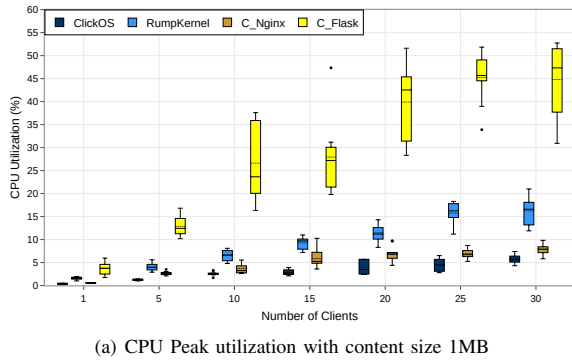


Fig. 7: Service assurance - CPU Peak

In Fig. 5 we observe that, in general, C_Flask outperforms other approaches in terms of *Total Delivery Time*, succeeding by ClickOS, RumpKernel, and C_Nginx. The same performance order is detected in the majority of cases, i.e., between 10 to 30 clients and for content sizes 5, 10 and 20MB. In the same figure, we notice an almost steady increase of *Total Delivery Time* with the number of client requests. For example, such increments for 5MB to 10MB or 10MB to 20MB range from 1.4 to 2.4 times, for all approaches.

From an analysis of *Total Delivery Time* and its constituting *Response* and *Download Times*, we see that: (i) *Response Time* is independent of content size and the number of client requests for ClickOS, RumpKernel and C_Nginx, which does not exceed value 0.06 sec; and (ii) C_Flask *Response Time* increases with content size and the number of client requests, but with an almost non-fluctuating behavior regarding *Download Time*, which is not the case in the other solutions. This attitude can be justified by the different approaches of Nginx and Flask web servers in terms of requests' handling. Practically, we use the simplistic web server integrated with Flask, which struggles to handle all client requests (i.e., also reflected a high *CPU Peak* value, as shown in Fig. 7), however leading to a better *Download Time* because it desynchronizes the parallel web requests. The standard deviation values for *Download* and *Response Time* range from 0.004 to 1.65 and 0.003 to 0.21, respectively, i.e., causing borderline differences between the third and fourth-placed approaches only.

As depicted in Fig. 6 in the majority of runs, C_Flask achieves the higher *Network Throughput* per client. We identify an almost resembled performance pattern among *Total Delivery time* and *Network Throughput*, i.e., due to the significance of the networking aspect in the considered application. The figure also indicates a decline of *Network Throughput* with client requests' number, but with higher fluctuations, compared to *Total Delivery Time's*, i.e., around 6% to 73% and 9% to 69%, for C_Nginx and ClickOS, respectively. The standard deviation of *Network Throughput* ranges from 0.07 to 3.83, but without impacting the above arguments.

In Fig. 7(a) and 7(b) we see that in most cases C_Nginx outmatches the other solutions in terms of lowest *CPU Peak*, i.e., at least by 8% to 10%. However, in all circumstances C_Flask faces the highest *CPU Peak* value and the greatest variability, highlighting that web service technology mattered more for *CPU Peak* than the virtualization technology, since both C_Nginx and C_Flask use containers. We also see a diverging behavior between unikernels and containers. For containers, *CPU Peak* values are increased until a particular value and then fluctuate, approximately, 48% to 50% and 7% to 8% for C_Flask and C_Nginx, respectively. Moreover, we see that *CPU Peak* increases more rapidly with a lower number of requests, as content size increases. Such attitude of C_Flask relates to its struggle to cope with high loads. Regarding unikernels, there is an increasing trend in *CPU Peak*, i.e., its increase ratio is gradually being reduced with

the number of requests. The standard deviation of *CPU Peak* fluctuates from 0.04 to 9.34, without affecting the performance differences in each individual case.

We now summarize our findings as follows: (i) C_Flask achieves the best performance in terms of *Total Delivery Time* by distributing the content up to 3.4 times faster and consuming 3.18 times more *Network Throughput*, compared to RumpKernel, but it faces up to 40.7% higher *CPU Peak* values, in contrast to ClickOS, which aspect is important, especially for resource-constraint edge cloud deployments; (ii) ClickOS and C_Nginx are descent options for the operation of considered content delivery service, with the former being, in general, characterized by the best *Total Delivery Time* and *Network Throughput*, while the latter by the lower *CPU Peak* values; (iii) unikernels, compared to containers, have a more stable increasing of CPU utilization with clients' number, e.g., offering better conditions for a relevant prediction algorithm; and (iv) the web service technology used matters, especially for CPU resource consumption, with a *CPU Peak* difference between C_Nginx and C_Flask reaching up to 43%, in favor of the former.

C. Scenario 3: Elasticity

After evaluating the behavior of heterogeneous virtualized web-based service entities on the basic edge cloud processes of resource allocation, removal and operation, we now assess their performance in a complete cloud environment, i.e., involving both core and cloud resources as well as scale up and scale down events triggered from predicted demands. We reproduce a cycled user pattern in the form of $\{100, 100, 100, 100, 100, 100, 100, 100, 20, 20, 20, \dots\}$, assuming a periodic rapid shift on client demands, alternating between high and low numbers of requests. Since the first two scenarios revealed that C_Flask achieves the lower *Total Delivery Time* in service operation but with a high *CPU Peak* and ClickOS very low *Resource Allocation Time* and *CPU Peak*, it comes natural to utilize the former technology at the core cloud and the latter at the edge. To evaluate the efficiency of this strategy, we contrast the performance of this setting against an equivalent one with C_Flask instead of ClickOS.

In practical terms, the core cloud solely serves the users up to points with rapid demand shifts. At these particular times, there is a rapid deployment of service nodes at the edge cloud, which are removed when the demand drops. Such elasticity events are guided from the assumption that each involved web server handles up to 20 requests, i.e., resources are being deployed for every 20 requests. For example, 100 clients should ideally utilize a C_Flask container at the core cloud and four edge nodes, i.e., either with ClickOS or C_Flask. However, this depends on the accuracy of the load prediction. The content size for the elasticity experiments is 5MB.

Our test environment assumes both an accurate and a moderate prediction mechanism, i.e., based on *Seasonal Autoregressive Integrated Moving Average (SARIMA)* and *Exponential Moving Average (EMA)*, respectively, so we also investigate the impact of demand prediction accuracy on both service fulfillment and assurance. Here, we do not focus on

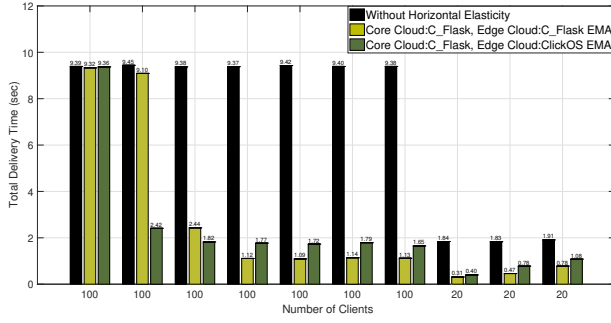
the prediction technology, rather than on its importance in the studied context. For the same reason, *SARIMA* is selected as a model that perfectly predicts the considered user pattern. According to our stepwise validation scheme based on the *root mean squared error (RMSE)*, the obtained parameters are *SARIMA(0,1,0)(0,1,0)10*. The first user pattern cycle does not produce a response of the system, since it is used for the training of prediction mechanisms. Finally, we set parameter n equal to 3 in the *EMA* formula, i.e., takes into account the last three inputs.

We have two sets of results, assuming both high (i.e., 100Mbps) and low (i.e., 10Mbps) bandwidth between *edge nodes* and *centralized node*, i.e., to evaluate the impact of network capabilities for service orchestration. Our goal is to assess both adequate and limited network resources, for the scale of our experiment, so we can get an indication whether the evolution of networking technology suffices (e.g., improving bandwidth) or strategies like the proposed should complement the latter, for the best outcome. In both cases, users request and download content over 200Mbps connections.

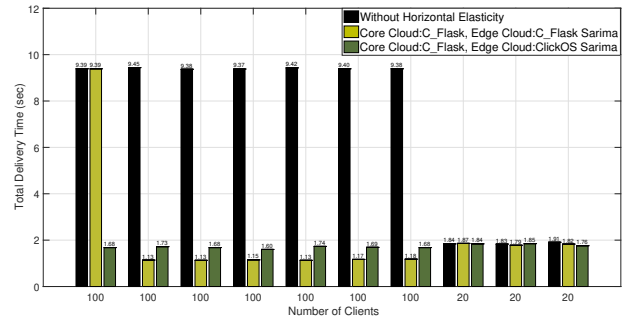
For clarity purposes, we depict the *CPU Infrastructure Utilization* figures with (i) indicative core and edge cloud nodes, since the visualization of all nodes produces a complex outcome, because the orchestration processes are not perfectly synchronized among the nodes; (ii) top and down subplots presenting the performance outcomes when using C_Flask and ClickOS at the edge nodes, respectively; and (iii) the cycled user pattern is visualized for one user batch sooner for *SARIMA* only (i.e., in Fig. 8(d) and 9(d)), so the impact of accurate prediction on *CPU Infrastructure Utilization* can be illustrated more clearly.

We start with the cluster of results assuming a high-bandwidth service orchestration. As expected, the involvement of edge cloud resources improves significantly the *Total Delivery Time*, i.e., as shown in Fig. 8(a) and 8(b). In the same figures, we observe that, independently of the load prediction strategy, ClickOS overall responds more rapidly compared to C_Flask, however C_Flask outmatches ClickOS in terms of *Total Delivery Time*, once the new edge cloud resources have been deployed. We also notice that *EMA*, compared to *SARIMA*, leads to further delays in the response of the system to the initial increased number of requests (switching from the 20 users of the training cycle to the 100 users of the regular cycle), i.e., takes one more round of users to respond. This first round is characterized by an equivalently high *Total Delivery Time* for the three approaches, indicating zero deployed edge cloud resources. The combination of accurate load prediction of *SARIMA* and rapid allocation capabilities of ClickOS leads to a very low *Total Delivery Time*, for all cases. It takes for C_Flask and *SARIMA* one round to respond to the high load, due to the less rapid resource allocation of the former, compared to ClickOS. Fig. 8(a) also highlights that *EMA* needs more rounds to predict load, leading to over-provisioning of resources in the last three user rounds, due to the associated gradual demobilization of resources.

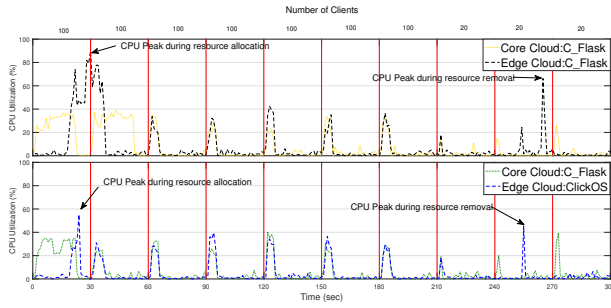
In Fig. 8(c) and 8(d) we notice the following: (i) ClickOS consumes less CPU resources compared to C_Flask for both



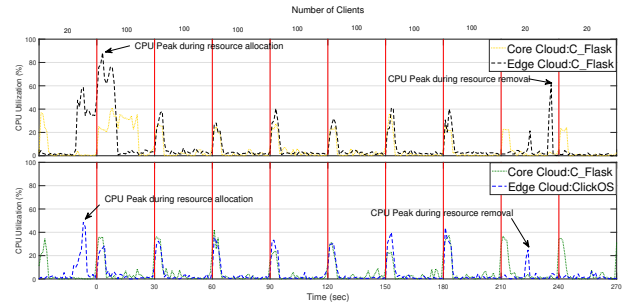
(a) Total Delivery Time with EMA-based load prediction



(b) Total Delivery Time with SARIMA-based load prediction

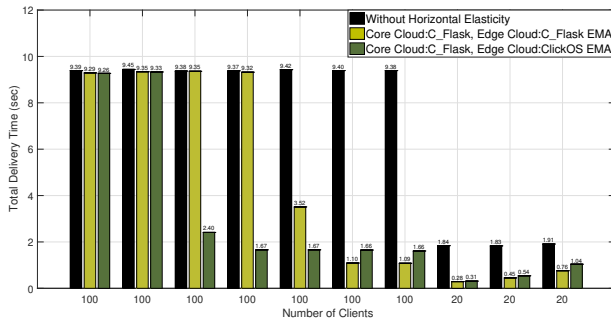


(c) CPU Infrastructure Utilization with EMA-based load prediction

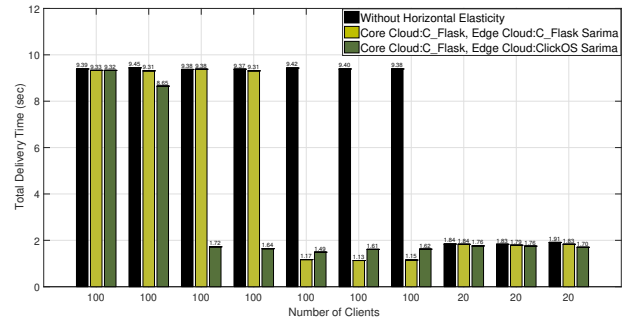


(d) CPU Infrastructure Utilization with SARIMA-based load prediction

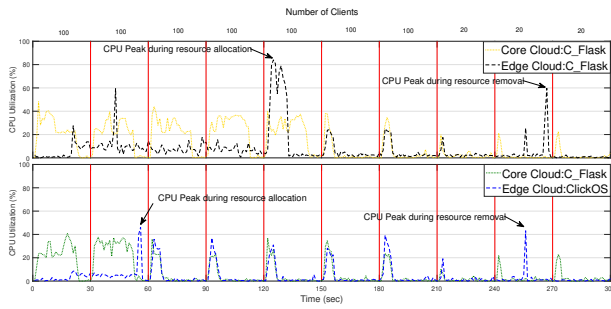
Fig. 8: Horizontal elasticity with 100Mbps bandwidth for service orchestration



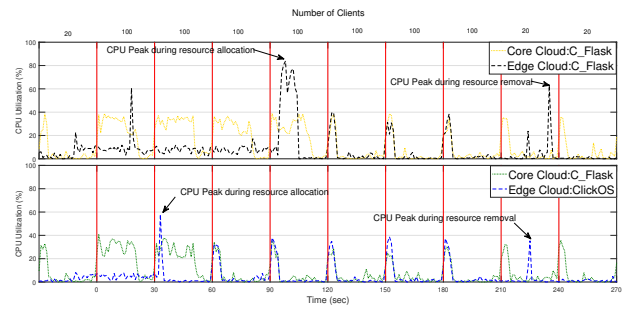
(a) Total Delivery Time with EMA-based load prediction



(b) Total Delivery Time with SARIMA-based load prediction



(c) CPU Infrastructure Utilization with EMA-based load prediction



(d) CPU Infrastructure Utilization with SARIMA-based load prediction

Fig. 9: Horizontal elasticity with 10Mbps bandwidth for service orchestration

allocation (i.e., about 30% and 35% less in *EMA* and *SARIMA*, respectively) and removal (i.e., 20% and 35% less in *EMA*

and *SARIMA*, respectively) of new resources, both in terms of *CPU Peak* and duration of *CPU involvement*; (ii) *EMA*

leads to a gradual response to the load changes, due to its slow prediction, while *SARIMA* responds rapidly, i.e., the latter allocates and removes resources almost 25 and 30 seconds sooner, corresponding to the first round of responses in the system, respectively; and (iii) the over-provisioning of resources due to inaccurate prediction from *EMA* is reflected to the lower *CPU Infrastructure Utilization* of core cloud towards the end of the corresponding figure curve, i.e., after the 210 sec of Fig. 8(c)

In the case of low bandwidth in service orchestration, there is an extension in image downloading time for the allocation of new virtual resources during the required vertical elasticity events, which is expected to impact the larger virtual entity sizes more. This behavior is manifested in the content delivery, since ClickOS is now allocated for the third user batch and C_Flask for the fifth, in the case of *EMA* (i.e., Fig. 9(a)). However, the accurate load prediction of *SARIMA* impacts marginally ClickOS, since it is ready to serve some users from the 2nd user batch (i.e., Fig. 9(b)). Consequently, the rapid booting up and small image size of ClickOS lead to a partial mitigation of performance issues caused by limited network resources in service orchestration.

An equivalent behavior can be seen in the *CPU Infrastructure Utilization* (i.e., Fig. 9(c) and 9(d)), where it takes more time for C_Flask to appear at the edge cloud and serve users, i.e., boots up almost 75 and 65 sec after ClickOS is up, for *EMA* and *SARIMA*, respectively. Such delay leads to an impact of around 30% to 40% on the core cloud's *CPU Utilization*, since resources are offloaded to the edge at a later stage. Things are better for resource removal, since there is no need to download images. Contrasting the impact of the two load prediction approaches on *CPU Infrastructure Utilization*, *SARIMA* leads to both sooner resource allocation and removal.

We now summarize our findings from the third scenario: (i) unikernels bring significant benefits in terms of responsiveness to rapid changes in the web load, e.g., ClickOS appears at the edge cloud and starts serving users up to 75 sec sooner than C_Flask; (ii) ClickOS is characterized by up to 40% and 35% lower *CPU Peak* values with respect to C_Flask, i.e., corresponding to the resource allocation and removal processes, respectively; (iii) inaccurate load prediction may lead to both over-provisioning or under-provisioning of resources, with *SARIMA* responding up to 25 and 30 sec earlier, in the respective instances of resource allocation and removal, compared to *EMA*, but such issues are more severe with containers, i.e., the system re-adjusts itself in a slower manner.

VI. DESIGN GUIDELINES FOR EDGE CLOUD SYSTEMS

The general deployment of computing facilities in telecom networks is introducing new challenges on their management, control and operation. Apart from interaction and integration with the underlying transport network [33], or the proper selection of the compute environment for deploying a given service [34], there are additional aspects of relevance to take into consideration, especially the possible virtualization approach [6] to follow, according to the network and service circumstances. This is even more evident with the general

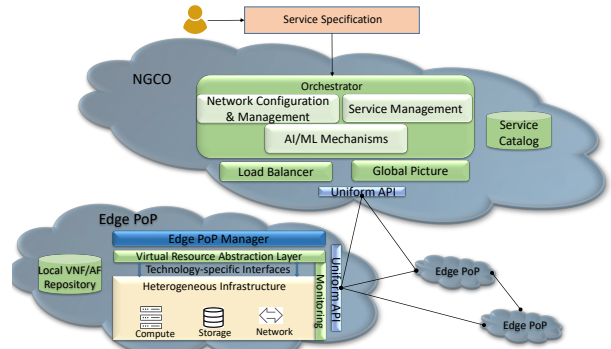


Fig. 10: Conceptual edge cloud orchestration platform

adoption of cloud-native approaches by different technological paradigms, in support of virtualized services [21] – [23].

In this context, we build-up on our research results towards providing design directions for 5G and beyond edge cloud infrastructures, which are aligned to the conceptual edge cloud platform illustrated in Fig. 10. The latter operates over multiple edge cloud Points of Presences (Edge PoPs), as well as a core cloud deployment (i.e., a next-generation central office - NGCO). A centralized orchestrator manages all network, compute and storage resources as well as service nodes, through Edge PoP managers that are deployed in every edge cloud. The traffic load is balanced between the Edge PoPs through a load balancer controlled from the same orchestrator. The platform supports alternative virtualization approaches and relevant packet processing optimization mechanisms, as well as implementations of particular application functions (AF) and virtual network functions (VNFs), i.e., exhibiting diverse performance trade-offs. The orchestrator receives service requirements through a Northern Interface (e.g., like in NECOS platform [35]), selects and configures the most appropriate virtualization technologies, mechanisms and service nodes, i.e., to deliver the service with the expressed performance requirements.

As a bottom line, we envisage the following capabilities for our conceptual platform and other relevant systems: (i) a virtual resource abstraction layer that supports multiple virtualization technologies and relevant features (e.g., containers and unikernels, eBPF, XDP [18], etc.); (ii) abstract well-design APIs translating uniform primitives to technology-specific compute and network control processes; (iii) intelligent optimization mechanisms selecting and configuring the most suitable virtualization technologies and service nodes to particular service requirements and network conditions, e.g., targeting ultra-low latency; (iv) novel Artificial Intelligence and Machine Learning (AI/ML) capabilities enabling automation processes, including for fluid elasticity [36] and efficient resource allocation, scaling, load balancing / workload assignment; (v) a monitoring abstraction that provides a global picture to a centralized orchestrator, i.e., of virtual resources, network conditions and client behavior, backed by accurate prediction or rapid detection mechanisms; (vi) lightweight and high-performing service orchestration processes and interactions of involved components, adaptable to expressed service requirements; (vii) edge PoP managers being responsible for

the control of virtualized network and application functions in the edge infrastructures, coordinated by the centralized orchestrator; (viii) local VNF/AF repositories hosting alternative implementations of particular virtual network or application functions with multiple virtualization technologies; and (ix) a service catalog enlisting available services, along with a description of the deployment, operational and performance characteristics of their constituting virtual network and application functions.

VII. CONCLUSIONS

In the context of this paper, we conducted an extensive comparative evaluation of alternative builds of virtualized exemplary web-services, involving both unikernels and containers, and identified that each option is characterized by particular performance trade-offs. Our experiments utilized a novel bespoke edge cloud experimentation infrastructure that considers virtualized service deployment, removal, operation, as well as both vertical and horizontal elasticity processes. We consolidated the gained insights from our realistic experiments and defined a conceptual edge cloud orchestration platform for 5G and beyond networks with its key design guidelines.

Our next steps include the design, implementation and extensive experimentation of the envisioned edge cloud orchestration platform (i.e., of Fig. 10), equipped with an architecture and mechanisms in adherence to the defined design guidelines.

Furthermore, we plan to extend our experimentation exercise and research towards (i) investigating the impact of different network or application services (e.g., video streaming), other hypervisors and heterogeneous server hardware; (ii) consider large-scale deployments over open-access testbeds (e.g., FED4FIRE+ FUTEBOL [37] or GENI Emulab [38]) and simulation tools (e.g., [39]); (iii) assessing NFV orchestration capabilities, such as those proposed in [40], [41], mixing and matching, as well as configuring, alternative virtualized service node builds with respect to performance goals / guarantees or dynamic changes in the conditions of the cloud or network environment; (iv) integrating and evaluating important virtual network optimization mechanisms, such as eBPF and XDP; and (v) applying appropriate prediction and rapid detection mechanisms on the workload behavior, including employing the change-point analysis algorithms introduced in [42].

Last but not least, we are especially interested to integrate the core ideas of this paper in the novel edge cloud infrastructure StarlingX [19], exploiting its complementary unique edge cloud orchestration capabilities.

ACKNOWLEDGMENTS

This work has received funding by the "GSRI FUNDING FOR THE YEAR 2019 (Award for the participation in competitive E.U. projects) [HORIZON 2020 - JOINT ACTION EU - BRAZIL] which is funded by the Ministry of Development and Investments- General Secretariat for Research and Innovation as a reward for the implementation of the project "Novel Enablers for Cloud Slicing (NECOS), HORI-

ZON 2020 (grant agr. nº 777067)" funded by the European Commission.

REFERENCES

- [1] T. Taleb *et al.*, "White Paper on 6G Networking," Jun 2020 Available: <https://www.6gchannel.com/>. Accessed on: September 2020.
- [2] J. F. Santos *et al.*, "Breaking Down Network Slicing: Hierarchical Orchestration of End-to-End Networks," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 16–22, Nov 2020.
- [3] A. Medeiros *et al.*, "End-to-end elasticity control of cloud-network slices," *Internet Technology Letters*, vol. 2, no. 4, p. e106, Jul 2019.
- [4] P. Heidari, Y. Lemieux, and A. Shami, "QoS Assurance with Light Virtualization - A Survey," in *Proc. IEEE International Conf. Cloud Computing Technology and Sciences*, Dec 2016, pp. 558–563.
- [5] A. Madhavapeddy and D. J. Scott, "Unikernels: Rise of the virtual library operating system," *Queue*, vol. 11, no. 11, pp. 30–44, Dec 2013.
- [6] 5GPPP Technology Board and 5G-IA's Trials Working Groups, white paper, "Edge Computing for 5G Networks," 2021. Available: <http://bscw.5g-ppp.eu/pub/bscw.cgi/d397473/EdgeComputingFor5GNetworks.pdf>. Accessed on: June 2021.
- [7] A. Madhavapeddy *et al.*, "Jitsu: Just-In-Time Summoning of Unikernels," in *in Proc. USENIX Symp. NSDI*, May 2015, pp. 559–573.
- [8] R. Behraves, E. Coronado, and R. Riggio, "Performance Evaluation on Virtualization Technologies for NFV Deployment in 5G Networks," in *IEEE Conf. on Network Softwarization (NetSoft)*, Jun 2019, pp. 24–29.
- [9] I. Mavridis and H. Karatza, "Lightweight Virtualization Approaches for Software-Defined Systems and Cloud Computing: An Evaluation of Unikernels and Containers," in *the 6th International Conf. on Software Defined Systems (SDS)*, Jun 2019, pp. 171–178.
- [10] M. Plauth, L. Feinbube, and A. Polze, "A Performance Evaluation of Lightweight Approaches to Virtualization," *Cloud Computing*, vol. 2017, p. 14, Feb 2017.
- [11] T. Kurek, "Unikernel Network Functions: A Journey Beyond the Containers," *IEEE Communications Magazine*, vol. 57, no. 12, pp. 15–19, 2019.
- [12] I. Briggs, M. Day, Y. Guo, P. Marheine, and E. Eide, "A Performance Evaluation of Unikernels," in *Technical Report*, 2014.
- [13] T. Goethals, M. Sebrechts, A. Atrey, B. Volckaert, and F. De Turck, "Unikernels vs Containers: An In-Depth Benchmarking Study in the context of Microservice Applications," in *in Proc. of 8th International Symposium on Cloud and Service Computing (SC2)*, Nov 2018, pp. 1–8.
- [14] B. Xavier, T. Ferreto, and L. Jersak, "Time provisioning Evaluation of KVM, Docker and Unikernels in a Cloud Platform," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CC-Grid)*, May 2016, pp. 277–280.
- [15] O. A. Ezenwigbo, J. Ramirez, G. Karthick, G. Mapp, and R. Trestian, "Exploring the Provision of Reliable Network Storage in Highly Mobile Environments," in *Proc. of the 13th International Conf. on Communications (COMM)*, Jun 2020, pp. 255–260.
- [16] J. B. Filipe, F. Meneses, A. Rehman, D. Corujo, and R. L. Aguiar, "A Performance Comparison of Containers and Unikernels for Reliable 5G Environments," in *in Proc. of IEEE International Conf. on the Design of Reliable Communication Networks (DRCN)*, Mar 2019, pp. 99–106.
- [17] V. Aggarwal and B. Thangaraju, "Performance Analysis of Virtualisation Technologies in NFV and Edge Deployments," in *Proc. of the IEEE International Conf. on Electronics, Computing and Communication Technologies (CONECT)*, July 2020, pp. 1–5.
- [18] M. A. Vieira, M. S. Castanho, R. D. Pacifico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, Feb 2020.
- [19] "StarlingX project," Available: <http://www.starlingx.io/>. Accessed on: September 2021.
- [20] A. Nasrallah, A. S. Thyagaruru, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2018.
- [21] ETSI GR NFV-IFA 029, "Network Functions Virtualisation (NFV) Release 3; Architecture; Report on the Enhancements of the NFV architecture towards "Cloud-native" and "PaaS"", V3.3.1 (2019-11). [Online], Available: https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/029/03.03.01_60/gr_NFV-IFA029v030301p.pdf. Accessed on: December 2020.

- [22] ETSI GR MEC 027, “Multi-access Edge Computing (MEC); Study on MEC support for alternative virtualization technologies”, V2.1.1 (2019-11), [Online], Available: https://www.etsi.org/deliver/etsi_gr/MEC/001_099/027/02.01.01_60/gr_MEC027v020101p.pdf, Accessed on: December 2020.
- [23] 5GPPP Software Network Working Group, white paper, “Cloud-Native and Verticals’ Services,” 2019, Available: https://5g-ppp.eu/wp-content/uploads/2019/09/5GPPP-Software-Network-WG-White-Paper-2019_FINAL.pdf, Accessed on: September 2020.
- [24] Docker, “The Docker Containerization Platform,” [Online], Available: <https://www.docker.com/>, Accessed on: September 2020.
- [25] Canonical Linux Containers (LXC), [Online], Available: <https://linuxcontainers.org>, Accessed on: September 2020.
- [26] A. Madhavapeddy *et al.*, “Turning Down the LAMP: Software Specialisation for the Cloud,” in *Proc. 2nd USENIX Conf. HotCloud*, vol. 10, pp. 11–11, Jun 2010.
- [27] J. Martins *et al.*, “ClickOS and the Art of Network Function Virtualization,” in *Proc. of 11th USENIX Symposium on NSDI 14*, Apr 2014, pp. 459–473.
- [28] J. Cormack, “The rump kernel: A tool for driver development and a toolkit for applications,” in *Proc. of the Asian BSD conf.(AsianBSDCon)*, Mar 2015.
- [29] A. Kivity *et al.*, “OSv—Optimizing the Operating System for Virtual Machines,” in *proc. of the USENIX Annual Technical Conf. (USENIX ATC '14)*, Jun 2014, pp. 61–72.
- [30] Softwarized Wireless Network, [Online], Available: <http://emulab.swn.uom.gr/>, Accessed on: December 2020.
- [31] Node-RED, [Online], Available: <https://nodered.org>, Accessed on: December 2020.
- [32] Langston Nashold and Rayan Krishnan, “Using LSTM and SARIMA Models to Forecast Cluster CPU Usage,” *CoRR*, 2020.
- [33] L. M. Contreras *et al.*, “Toward cloud-ready transport networks,” *IEEE Communications Magazine*, vol. 50, no. 9, pp. 48–55, Sep 2012.
- [34] L. M. Contreras, J. Baliosian, P. Martínez-Julia, and J. Serrat, “Computing at the Edge: But, what Edge?” in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, Apr 2020, pp. 1–9.
- [35] S. Clayman, A. Neto, F. Verdi, S. Correa, S. Sampaio, I. Sakellariou, L. Mamatas, R. Pasquini, K. Cardoso, F. Tusa *et al.*, “The necos approach to end-to-end cloud-network slicing as a service,” *IEEE Communications Magazine*, vol. 59, no. 3, pp. 91–97, 2021.
- [36] P. Valsamas, S. Skaperas, and L. Mamatas, “Elastic Content Distribution Based on Unikernels and Change-Point Analysis,” in *Proc. 24th Eur. Wireless Conf.(EW)*, May 2018, pp. 1–7.
- [37] C. Both *et al.*, “Futebol control framework: Enabling experimentation in convergent optical, wireless, and cloud infrastructures,” *IEEE Communications Magazine*, vol. 57, no. 10, pp. 56–62, Oct 2019.
- [38] P. Valsamas, I. Sakellariou, S. Petridou, and L. Mamatas, “A Multi-domain Experimentation Environment for 5G Media Verticals,” in *IEEE INFOCOM Workshop on Computer and Networking Experimental Research using Testbeds (CNERT)*, Apr 2019, pp. 461–466.
- [39] T. Goyal, A. Singh, and A. Agrawal, “Cloudsim: simulator for cloud computing infrastructure and modeling,” *Procedia Engineering*, vol. 38, pp. 3566–3572, Jan 2012, International Conference on Modelling Optimization and Computing. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S187705812023259>
- [40] M. S. Castanho, C. K. Dominicini, R. S. Villacça, M. Martinello, and R. M. Ribeiro, “Phantomsfc: A fully virtualized and agnostic service function chaining architecture,” in *IEEE Symposium on Computers and Communications (ISCC)*, Jun 2018, pp. 354–359.
- [41] C. K. Dominicini, G. L. Vassoler, L. F. Meneses, R. S. Villaca, M. R. Ribeiro, and M. Martinello, “Virtphy: Fully programmable nfv orchestration architecture for edge data centers,” in *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 817–830, Sep 2017.
- [42] S. Skaperas, L. Mamatas, and A. Chorti, “Real-time video content popularity detection based on mean change point analysis,” *IEEE Access*, vol. 7, pp. 142 246–142 260, 2019.



Polychronis Valsamas (S'19) pursues his Ph.D. in the area of Cloud Computing and the Network Edge at the University of Macedonia, Thessaloniki, Greece. He holds a M.Sc. Degree in System Engineering and Management from the Democritus University of Thrace, Xanthi, Greece. In his MSc thesis he experimented with Lightweight Clouds based on the unikernel Technologies. He also holds a BSc Diploma from the Department of Informatics and Telecommunications Engineering University of Western Macedonia, Kozani, Greece.



Lefteris Mamatas (S'04, M'08) received the diploma and Ph.D. degree from the Department of Electrical and Computer Engineering, Democritus University of Thrace, Greece. He is an Assistant Professor in the Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece. His research interests lie in the areas of Software-Defined Networks, Internet of Things, 5G Networks, and Multi-Access Edge Computing. He has published more than 60 papers in international journals and conferences.



Luis Miguel Contreras received the M.Sc. degree in telecommunications engineering from the Universidad Politecnica de Madrid (1997) and the M.Sc. in telematics from the Universidad Carlos III de Madrid (UC3M) (2010). In 1997 he joined Alcatel Spain, working in both wireless and fixed networks. In 2006 he joined the Network Planning department of Orange Spain (France Telecom group), working on IP backbone planning. Since 2011 he has been with Telefonica Global CTO, working on scalable networks and their interaction with cloud and distributed services, and participating on several FP7 projects. He works toward the Ph.D. degree in the Telematics Department of UC3M.