Relocation of container-based services in a MEC-NFV orchestrated environment

Cristina Bernad[1 [0000−0001−9537−415X]], Vojdan Kjorveziroski,[2 [0000−0003−0419−4300]], Pedro Roig,[1 [0000−0002−8391−8946]], Salvador Alcaraz,[1 [0000−0003−3701−5583]], Katja Gilly,[1 [0000−0002−8985−06395]], and Sonja Filiposka[2 [0000−0003−0034−2855]]

[1] Dept. of Computer Engineering, Miguel Hernández University, Elche (Alicante), Spain, {cbernad,proig,salcaraz,katya}@umh.es
[2] Fac. of Computer Science and Engineering, Ss. Cyril and Methodius University, Skopje, North Macedonia, {vojdan.kjorveziroski,sonja.filiposka}@finki.ukim.mk

Abstract. With the rapid growth of real-time next-generation mobile services, it has become necessary to work towards holistic orchestration of the benefits promised with edge computing based on bringing the computing infrastructure closer to the end user. While the concept of Multi-access Edge Computing (MEC) integrated with Network Function Virtualisation (NFV) is being standardised, there is still a lot of work to be done to orchestrate the relocation of edge applications integrated in 5G and beyond systems in a smooth and efficient manner. In this paper, we document the current status of the transparent relocation of edge services in an experimentally deployed MEC-NFV environment based on OSM. Working towards gathering monitoring training datasets necessary for the development of proactive MEC application orchestrators that will implement seamless follow-me behaviour for MEC services, we provide benchmark results for the service downtime of three potential MEC services hosted in lightweight containers. Our analysis of results shows that containers exhibit improved performance over that of virtual machines, but there are still some issues that require improvement in both the orchestration implementation as well at the relocation process for containers.

Keywords: Edge Computing · Optimisation · Standardisation · Migration · Mobile nodes · Containers.

1 Introduction

Dynamic online services have an important role in people's daily lives worldwide. With the promised specifications of 5G [1], which include ultra-low latency (less than 1 ms), high data rates (over 10 Gbps), and high reliability (99.999%), the design and implementation of real-time services for mobile users that make use of the increased bandwidth and reduced latency times has been accelerated, promoting applications such as 4K video, autonomous cars, augmented reality,

telemedicine, or IoT devices. This explosion of services with a wide variation in flexible resource requirements in terms of networking and computing needs to be addressed by taking advantage of several complementary technologies.

The self-organisation of networking systems emerges as an important challenge that requires to be taken into consideration so as to provide the network with the necessary capabilities to readjust resource management and network configurations in real-time based on the highly changing service demands. Technological advances such as Software Defined Networking (SDN) technologies, which define and adapt network topologies programmatically and Network Function Virtualisation (NFV), which enables the virtualisation of network services and functions such as firewalls, routers, etc; provide the ability to build agile networks with on-the-fly configurations [2].

To fully support the next-generation users of highly demanding applications, the computation facilities need to be brought closer to data sources thus reducing the user-perceived latency. In other words, 5G and beyond services need to be integrated with edge computing solutions such as the Multi-access Edge Computing (MEC) initiative promoted by the European Telecommunications Standards Institute (ETSI), which has spent years working on standardising the edge architecture for mobile users (MEC) [3]. Moreover, the union of 5G and MEC facilitates an optimal result when dealing with the requirements of real-time applications (high bandwidth, ultra-low delay and intense computational effort), requirements that the cloud cannot fully cover. Recognising that the marriage of MEC and 5G will pave the way for a holistic approach to developing a unified system based on mutual benefits, ETSI has released a new version of the MEC architecture standardisation that focuses on implementing the MEC system as a part of an NFV architecture, thus effectively incorporating MEC into 5G [4]. This view provides a way to manage the lifecycle of MEC applications alongside other virtual network functions (VNFs), while recognising the need to orchestrate the MEC services in a specific way that enables the implementation of placement and migration policies based on the quality of service (QoS) and user location. The overall VNF orchestrator in the system is managed by an NFV Management and Orchestration (MANO) orchestrator [5].

Using servers co-located with 5G base stations at the edge, as close to the end user as possible, MEC applications are released from the computational load, delay is reduced and bandwidth is gained. To ensure a sustained high-performing edge system, efficient implementation of the edge facilities and their highly dynamic resource management is a must. The increased complexity in this scenario is due to the fact that mobility is implicit in online mobile services. Thus, continuous ultra-low latency requires that the MEC application 'follows' the user's trajectory [6]. Therefore, taking into consideration a changing ecosystem, MEC applications may be instantiated in the currently optimal computing node, but as users move to a different geographical area, i.e. the user is being handed off from one 5G base station to another, the allocated MEC service needs to be moved to a 'closer' computing node. It is strictly necessary for the MEC-NFV system to be able to manage and orchestrate this 'follow me' practice,

that permits MEC applications to be migrated between geographically dispersed computing nodes.

The brains for this operation in the MEC-NFV architecture fall on the shoulders of the MEC application orchestrator [7]. This component is vital when it comes to the decision-making on optimal placement of new MEC services as well as maintaining high QoS during the MEC service use by smartly migrating the MEC service using the follow-me paradigm. The decisions made by the MEC orchestrator can trigger adjustments in the MEC-NFV system via the NFV orchestrator. A proactive decision-making implementation of the MEC orchestrator can take advantage of the current and past knowledge about the system to forecast future service migrations and thus improve the overall system performance. This is the reason whythe attention in MEC orchestration and decision-making has been focused on proactive approaches, mostly based on Artificial Intelligence (AI) [8]. However, to be able to take full advantage of the potential of AI forecasting the model needs to be fed with a lot of information regarding the users and services status.

To address the challenge of online services live migrations in a MEC architecture, we have developed a testbed environment based on ETSI MEC-NFV [4] using the Open Source MANO (OSM) orchestrator. Considering the functions that each element of the architecture must perform [9], we have carried out tests with different types of potential MEC applications according to their computational load. Traditionally in the ETSI MEC ecosystems, virtual machines (VMs) are considered as the virtualisation technology used to host MEC services. Observing the increasing use of containers in virtual computing environments, we have developed an experimental MEC implementation using containers to host MEC services aiming to verify the behaviour of the MEC orchestrator when performing on-demand service migrations. This has also enabled us to analyse the current possibilities to implement seamless container relocation in a MEC-NFV architecture implemented using OSM as the NFV orchestrator. While application instantiation benchmark results exist using the old version of OSM, to the best of the authors' knowledge, there is no study that provides benchmark results regarding the relocation of standalone containers in an OSM-based MEC-NFV experimental environment.

We summarise the contributions of this paper as follows:

- investigate the ability to implement 'follow-me' relocation functionality in the current OSM release while using standalone containers to host MEC services,
- comparative performance analysis of migrating Kubernetes containers and Microstack virtual machines in an MEC-NFV environment,
- gather benchmark migration data that can be used for training proactive MEC orchestrator resource management algorithms.

The rest of the document is structured as follows: In section 2, we address the background and motivation of our tests. We analyse the current state of using OSM to implement a MEC-NFV system based on standalone containers that will support seamless follow-me behaviour in section 3. In section 4 we discuss

the developed experimental environment. In section 5 we present the results obtained and finally we conclude the article and comment on future work.

2 Background and Motivation

The problem of optimal placement of MEC services that needs to be solved by the MEC orchestrator has been a very active field of research in recent years. Different approaches have been proposed to correctly identify the 'correct' node where the MEC service should be instantiated so as to achieve maximum performance. In [10] the authors propose a Markov approximation algorithm to optimise the placement of shared VMs taking into account the price of implementing MEC systems. In [11] they focus on minimising the average response time using a latency-aware heuristic placement algorithm for VM placement, while the topic of energy-aware resource placement using trees social relations algorithm is the topic of interest in [12].

In contrast to the typical reactive approach, where the optimal placement is decided upon receiving a service request, lately, there is a growing number of proactive approaches that are attempting to forecast an optimal placement before the actual request comes in. Most of these approaches are based on the implementation of Machine Learning (ML) techniques such as [13] and [14], including deep learning [15]. In addition, recent research has also focused on the need to relocate the once-instantiated MEC services in order to preserve optimal QoS. The study in [16] proposes a Deep Reinforcement Learning to estimate optimal policy that jointly minimises migration cost, transaction cost, and consumed energy. It also provides a very good overview of the different approaches to MEC service migration both conventional and learning-based.

Typically, the performance of the proposed approaches is based on numerical analysis or simulation frameworks that enable the analysis of what-if scenarios as is the case with [16] for one example. To ensure that the perceived performance will be attainable in real-life implementations, it is essential that the work is also tried and tested in an experimental testbed that will provide the opportunity to ascertain how all components of the 5G-MEC system are orchestrated by the MEC orchestrator. For these purposes, a well-defined, validated and benchmarked testbed needs to be established, which not only incurs cost and resources but also requires experience in setting up and managing virtualised infrastructures and MANO solutions. Another problem that arises is that the current development of MEC orchestration functionalities mostly focuses on using VMs to host MEC services, while real-life implementations are starting to move away from VMs and turn towards containers as their lightweight cloud-native counterparts due to optimised performances and pricing.

To help close this gap of MEC orchestration analysis in an experimental environment, researchers have proposed new edge orchestration architectures such as [17] that serves as a MEC platform where MEC services can be hosted in containers and is compliant with the main ETSI MEC architecture. The authors in [18] propose their own orchestration architecture that is loosely based

on ETSI NFV and MEC aiming to provide multi-layered orchestration. However, having in mind that the ETSI NFV architecture is closely followed in the 5G implementations, and the ETSI MEC-NFV [4] provides a multi-layer orchestration for this scenario, using open-source solutions that implement the NFV MANO functionalities will enable building an experimental scenario that can be very close to real-life implementations. Towards this goal, the authors in [19] validate and benchmark a test-bed implementation of the ETSI-compliant Open Source MANO (OSM) [20] in combination with Kubernetes containers as the main virtualisation technology. The authors in this work focus on the placement of container-based network functions as the newly available feature in OSM release 7. OSM is one of the most popular open source NFV MANO solutions [21] and is therefore the solution of choice when building experimental testbeds as is the case with [22] and [23]. In both works the focus is on relocation of containers managed by OSM and hosted inside an OpenStack virtual infrastructure. While [23] uses OpenNESS as the edge management platform for the containers inside OpenStack, [22] builds an additional layer with the MEC orchestrator in accordance with the ETSI MEC-NFV proposed architecture. The focus of both is the implementation of application context switching for container migration and its incurring delay in the system that has been found to be increasing linearly with its size.

As the current research focus is put mostly on placement-only for MEC services together combined with experimental environments that are implemented with containers inside VMs, we aim our research on the topic of validating and benchmarking relocation of containers deployed using a standalone Kubernetes platform in an ETSI MEC-NFV environment using OSM. Thus we complement and build upon the work discussed previously while using a newer release of OSM compared to previous efforts. By analysing the latency incurred when relocating several different MEC applications as both VMs and containers, we are able to produce additional monitoring data that can be of further use to the deep learning MEC orchestration algorithms that currently mostly focus on data related to the user and base station.

3 Container relocation in a MEC-NFV environment using OSM

The ETSI's MEC initiative is promoting a standardisation effort that periodically provides enhanced versions of its open environment framework and reference architecture for edge computing in mobile environments. The latest release [3] includes a variant for MEC deployment in an NFV environment given in Figure 1 wherein the role of the MEC Orchestrator which is the management heart of the MEC ecosystem is split between the 'Mobile Edge Application Orchestrator' (MEAO) and the 'Network Functions Virtualisation Orchestrator' (NFVO).

This division of functionalities enables the integration of the ETSI NFV high-level functional architectural framework and design philosophy of virtualised net-

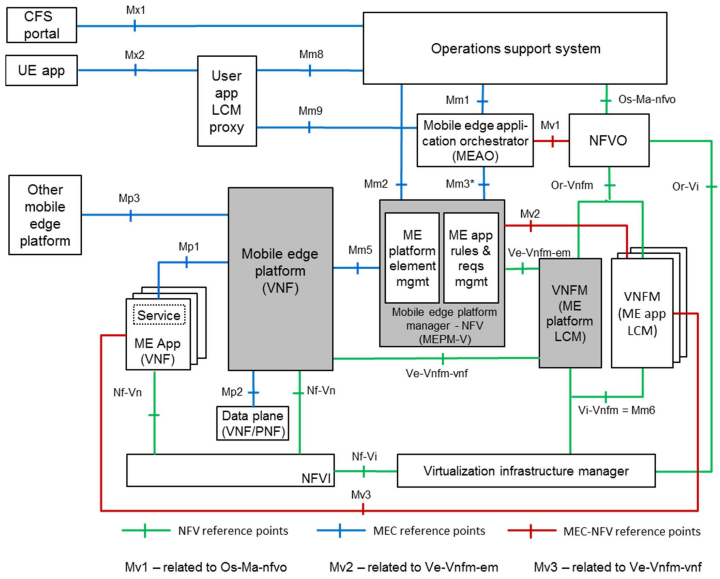


Fig. 1: NFV-based MEC generic architecture [3].

work functions in the edge reference model. When handling MEC applications, the MEAO module is the high-level orchestrator that decides when and where to put/move a virtualised instance based on user information, monitoring information, available resources and topology. The actual actions of management of the lifecycle of a MEC application are left to the NFVO which translates the MEAO orchestration decisions to Virtual Network Functions (VNF) language and executes them. The ETSI NFV special interest group has issued a standard NFV Management and Orchestration (MANO) framework [5] that covers the functionality of the NFVO module defined in the MEC-NFV architecture and with OSM developed as an open source implementation of this framework [20].

Within the MEC-NFV architecture from the perspective of NFVO the MEC application is considered as just another type of Virtual Network Function (VNF). The typical implementation of VNFs supported by existing MANO frameworks is in the form of a VM. In recent years, interest in containers as an alternative virtualisation technology has grown significantly. ETSI is putting more effort into integrating CNFs (Containerised Network Functions) in the reference system [24] as the range of implemented services as containers is constantly increasing. Applications running in containers can be deployed faster by downloading from many public repositories and simply executing, which not only saves time for the development teams but also enables a straightforward expansion of the MEC service portfolio with a large number of third-party applications.

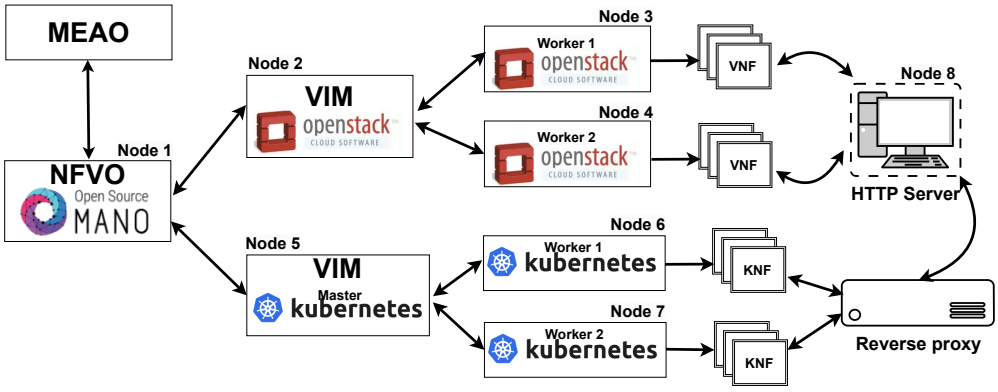


Fig. 2: Configuration used in our experimental deployment from Kubernetes perspective

The key difference between VMs and containers is that VMs virtualise the entire machine down to the hardware (hardware-level virtualisation), and containers only virtualise the software layers above the operating system (OS) level. In hardware-level virtualisation, the hypervisor (also called virtual machine monitor) emulates server hardware resources to the different deployed VMs. Each VM runs its OS and applications in a way that several instances of virtualised applications effectively share the same physical server. On the other hand, virtualisation at the OS encapsulates dependencies such as libraries and OS processes to create containers. Containers are deployed using a lightweight virtualisation layer that leads to obtaining an improved performance at a lower cost. Both virtualisation systems can be deployed in the 5G and MEC ecosystem.

In accordance with this move towards containers, in its release 7, the feature to manage CNFs has been added to OSM. However, even with the regular new releases of OSM with the current latest being 14, it still doesn't offer full support for CNFs. Instead, OSM MEC applications are implemented as VNFs considering the MEC-NFV architecture in Figure 1. The use of containers in the OSM context is still not straightforward and mostly stays in the same status as in release 7 [19]. From an MEAO point of view, this creates a complicated situation, as the MEC orchestrator expects to be able to fully control the location of placement and relocation of containers and get real-time info about their status. In this section, we aim to describe how container relocation can be implemented in OSM and what are the lessons learnt while implementing our experimental testbed.

In alignment with the ETSI NFV MANO specification, the management of the underlying virtualised infrastructure is performed by the Virtualisation Infrastructure Manager (VIM) that prepares the infrastructure to boot the soft-

ware image and reports information about it. It also is the one that should execute a relocation of any application when commanded by the NFVO.

In the OSM case, if the VIM is implemented based on VMs, using OpenStack or MicroStack for managing VMs, for example, then the addition of the MEAO in the overall system is quite straightforward. OSM supports the ability to control the destination node for a network service placement, and it offers the ability to migrate a service from the source node to the destination node, while at the same time reporting on the current status and progress. This means that the MEAO components only need to call the OSM Northbound API to be able to effectuate the decision made using its smart follow-me algorithms.

When it comes to using containers, the situation is much more complicated. As already mentioned, OSM supports network functions based on Kubernetes in the form of Kubernetes-based VNFs (KNFs) since version 7. Ever since the features regarding managing KNFs with OSM have not improved much. To be able to run KNFs, required to create a Kubernetes cluster and link it to a VIM network [25] so that it can be used for MEC services deployment via OSM. In this way, the Kubernetes cluster should behave as expected even when the pod management commands come from a higher entity such as OSM.

To build the test environment, we have followed the requirements described in the OSM documentation that include installing a load balancer (used for automated instantiation of containers based on current performance) and the storage class that includes the non-SQL database system MongoDB [26]. It is important to note that there are two different options to connect OSM with a Kubernetes (K8s) cluster:

- the cluster can be deployed inside a VM that is running in the OSM's registered VIM, where all VMs are connected to the VIM network; or
- a standalone K8s cluster (bare-metal or virtualised) can be physically connected to the VIM network.

All previous research related to the relocation of containers that are known to the authors is implemented using the first option: inside a VM. The reason behind this is that currently, this is the only way to use the OSM Northbound API to control the placement of the KNFs indirectly, by controlling the VMs on which the worker nodes are placed, requiring a distributed work nodes approach as in [22]. This scenario is however not desirable when it comes to performances and, more importantly, direct use and management of containers.

In this paper, we opted to choose the second configuration of standalone containers for our environment. As we chose to implement the K8s cluster outside the OSM's VIM, we have defined a controller node and two separate worker nodes that implement Network Services (NS) using Kubernetes containers. In this way, we can gain the maximum performance of the worker nodes unlike [19] wherein the worker nodes are co-located with the controller which is not recommended for production environments. The details of setting up standalone containers are fully documented in [19]. In this text, we only highlight the most important steps to emphasise that the need for clear distinction and improved handling of CNFs still persists.

The definition of an edge service in OSM is provided in a network service descriptor (NSD) that is defined in YAML and includes a description of the service format and requirements. Although similar to what is used to deploy VMs, it needs to be adapted for use with Kubernetes containers and linked to the external Kubernetes cluster. We provide the adapted NSD that can then be deployed as a KNF in Listing 3(a). As the structure of the definition is exactly the same as the one used to define network services for VMs, the container-based NSD is relatively simple to develop. The adapted NSD defines a virtual link descriptor (VLD) that is provided where it can be observed that the `'ClusterIP'` is the label used to identify the OSM's internal IP address that connects to the Kubernetes cluster. This label is also defined in the Kubernetes cluster as an external IP address. The ClusterIP service, default in Kubernetes, offers specific communication within the cluster, with access restricted to other applications or internal services. To boost Internet access, the Kubernetes proxy is used. In contrast, Ingress Controller is not classified as some kind of service in itself, but rather as a vital substance that sits before numerous services, acting as the highest passing point for our cluster. This device is especially cost-effective when you want to present different services over the same IP address, offering a perfect solution for test and development situations. We chose to use Nginx, which is one of the most common and versatile solutions for this purpose.

Finally, Listing 3(b) defines the adapted virtual network function descriptor (VNFD) that is now used as a KNFD with a single Kubernetes deployment unit (KDU) that links to the K8s repository in OSM using a 'helm-chart' from where the image of the application is downloaded to be immediately instantiated in a Kubernetes pod. A K8s repository is defined in OSM where a large number of edge applications can be directly downloaded and instantiated. Note that typically for VMs the hardware resources that constitute the VNF are defined in the VNFD wherein you can define the location of the VNF. On the other hand, in KNFs this is not possible.

In essence, for the standalone K8s option in OSM this translates to the inability to use the OSM Northbound API to control the placement and relocation of the KNFs, leaving this to the Kubernetes controller that will place the containers, re-instantiate them in another pod or scale them based on the monitoring feedback received. This makes the development of a MEAO that will control standalone KNFs via OSM not possible at the moment.

Another important difference when considering containers instead of VMs, is that the concept of container migration is quite different and significantly more difficult to perform [27]. While for VMs the process of live migration is fully implemented and documented in many available VIMs today, to migrate a service for containers such as Kubernetes, it is required to first move a pod from one edge node to another by creating an identical container on the destination node and then deleting the container on the source node (leaving it to the SDN implementation to seamlessly move the network connection for the user). Because of the complexities when it comes to container migration, there is no single command to migrate a container available in OSM nor in the underly-

```
vld:
- name: external
  vim-network-name: external
  additionalParamsForVnf:
- member-vnf-index: lang_detec_knf
  additionalParamsForKdu:
  - kdu_name: lang_detec_knf
    additionalParams:
      ingress:
        enabled: true
        className: nginx
        url: lang-detec.test.local
      service:
        type: ClusterIP
      name: knf_packets_received
      performance-metric: packets_received
```

a) Additional parameters for the NSD defined in OSM

```
vnfd:
  description: KNF with single KDU using helm-chart
  df:
  - id: default-df
  ext-cpd:
  - id: mgmt-ext
    k8s-cluster-net: external
  id: lang_detec_knf
  k8s-cluster:
    nets:
    - id: external
  kdu:
  - name: lang_detec_knf
  helm-chart: lang-detect-KNF/flask-lang-detect
  mgmt-cp: mgmt-ext
  product-name: lang_detec_knf
  provider: test
  version: 1.0
```

b) KNFD links with the K8s cluster and repository

Fig. 3: YAML definitions of container-based network services in OSM

ing Kubernetes infrastructure as is the case with VMs. As a showcase example of these complexities, in [28] the challenges of performing a live migration of containers made up of one or more pods (the minimum unit in Kubernetes) is discussed. The authors focus their efforts on extending the capabilities of K8s to perform a live migration using the CRIU tool that provides the ability to transfer the container checkpoint states on the destination node. However, as the source pod is deleted and another one is created on the destination during the migration, it is not possible to maintain smooth IP/TCP connections as the pod IP changes on the destination host. This is where SDN comes into play to ensure transparent networking from the user perspective. In addition, note that if the MEC services are stateful and require context switching during the container migration procedure then this must be performed manually by following the process described in [22].

To circumvent the problem of no existing readily available Northbound API in OSM for fine control of the KNF placement for standalone containers, we have developed an additional software module that relocates an existing MEC service from the source to the destination worker node. This module is used by our MEAO to be able to implement the follow-me behaviour on-demand. In order to be able to monitor the relocation process (as this action is not reflected in the MEC service status in OSM) we have also implemented an HTTP benchmark that can inform us of the down/up status of the container in question. In this way, the MEAO can get its information regarding the process of relocation.

Overall the lessons learnt from our investigation and implementation can be summarised as follows:

- support for CNFs has not advanced much since its first release in version 7, including the use of VNFD
- fine grain control for the positioning and relocation of VMs via GUI and Northbound API is available
- positioning control of containers in a standalone scenario is not available, load balancing is used, service status is reported
- relocation control of containers in a standalone scenario is not available, must be custom developed, the service status is not recognised and if needed must also be custom implemented
- interaction between OSM and KNFs can present interoperability challenges. Components must integrate properly which usually requires extra effort, especially when it comes to choosing compatible versions
- lack of detailed error messages can significantly complicate the identification and resolution of problems related to the deployment of KNFs in the container infrastructure

In the next section, we provide the technical details of our experimental setup and the MEC applications used for benchmarking.

4 MEC-NFV testbed setup details

To aid in the development of the tests and facilitate their performance testing, a set of 8 hosts was used, all connected to the same physical switch, providing 1Gbps connectivity. The nodes were placed on the same local network to eliminate any unforeseen impacts that could jeopardise the validity of the results, such as jitter or packet drops. No other workloads or tasks were running on them, other than those required for testing.

One host was dedicated to hosting OSM which assumes the NFVO role. The MEAO has been implemented as a separate component that performs API calls to start, migrate, and terminate edge services, and receives status and monitoring information. Two independent VIMs have been implemented to compare VM and container implementations: first VIM is set in a second node using a multi-node Microstack that controls two separate compute nodes in two additional nodes, and the second dummy VIM is connected to standalone a Kubernetes

Name	Role	Software version	GB RAM	CPU cores
Node 1	OSM	OSM 12	16	8
Node 2	MicroStack control plane	microstack (beta) ussuri	8	4
Node 3-4	2 x MicroStack compute nodes	microstack (beta) ussuri	8	4
Node 5	Kubernetes master	v1.22.17+k3s	8	4
Node 6-7	2 x Kubernetes workers	v1.22.17+k3s	8	4
Node 8	HTTP server	Ubuntu 20.04.3 LTS	8	4

Table 1: Host role and Hardware specifications for the experimental setup.

master with two Kubernetes nodes in three separate nodes as in Figure 2. The external Kubernetes cluster was deployed using the K3s lightweight Kubernetes distribution since existing research has shown that it is the preferred option for resource-constrained environments, such as the edge [29].

For the VMs case, all actions including monitoring are performed using the OSM Northbound AP. In the case of containers, an additional software layer has been developed to perform the process of container relocation and receive status information. The last node was dedicated to the configuration of an HTTP server to where we direct the HTTP requests for the tests in order to obtain the external performance measurements of the relocation operation.

Table 1 summarises the software versions and hardware specifications used in the complete experimental setup. Note that the software versions of the Kubernetes cluster and the OSM must be compatible for the system to work as a whole. In other words, careful combing through the documentation must be done to ensure that the version of your Kubernetes cluster is supported by the OSM release in question.

In order to obtain a sufficiently wide range of results, we have considered different application sizes to be relocated in a VM and container environment that we identify as small (S), medium (M) and large (L). All chosen applications are such that they do not require context transfer during the migration process so as to be able to measure the performance in a clean, fast scenario. The price of context switching can then be added according to the findings in [22] as the particular Kubernetes setup does not have a significant impact on the context switching functionality. Table 2 summarises the chosen applications for the VMs and containers scenarios. Note that the applications can not be the same for both cases as the size of the same application for VM and container differs greatly. The goal of this performance analysis is on the other hand to test different types of applications and check how their sizes impact the service downtime due to relocation. Care has been taken that all chosen container-based MEC apps expose a REST API that can be called via the HTTP server so as to implement the external service monitoring system using the hey benchmark.

The procedure to install the chosen apps starts with adding the repository to OSM. VNF, KNF and NS packages are developed and uploaded to OSM.

	Container MEC Applications		VM MEC Applications	
Designation	**App. Type**	**Compressed Img. Size**	**App. Type**	**Image Size**
S	Owntracks location tracker	20.2MB	Cirros	12.13 MB
M	Flask language detection	178MB	Xenial server	300.75 MB
L	R studio plumber	996 MB	bitnami ELK	1.17 GB

Table 2: Example MEC applications in three sizes S, M and L

Then, OSM is called to start the deployment of a VNF or KNF depending on the scenario, allowing the VIM control plane to decide on which node the service will be instantiated based on load balancing. If an error occurs during the deployment of a KNF, it is essential to have precise and specific information about the problem in order to take corrective measures. However, in some cases, the error messages provided by OSM may be generic or lack sufficient detail to understand the root cause of the problem.

5 Results

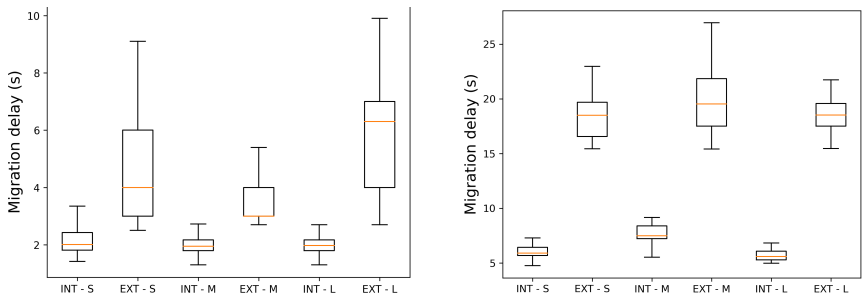
Using the prepared experimental testbed, we have carried out a number of tests aimed at monitoring the performance of relocation for both VMs and containers.

In the VM migration scenarios, the process of live migration is triggered by the MEAO with simple API calls to OSM. The measurements done in this case are based on the monitoring and status information available in OSM, we also refer to these as external measurements. These measurements have been augmented with the corresponding information that can be found in the NOVA module and instance logs of Microstack. We refer to the latter as internal measurements As our investigations have shown that OSM uses a period pull mechanism to query the status of a started migration operation with a default period of 10s, in order to obtain more granular and precise results we have changed this behaviour by enabling pull requests every 1s.

To monitor the performance of the relocation operation in containers we have carried out two types of tests

- external - we take measurements using an Apache HTTP server in order to mimic the OSM pulling for monitoring and obtain the latencies as they would be perceived by the NFVO. We measure how long a service is unavailable during its migration by continuously sending requests from an HTTP benchmark to the container
- internal - we consider measurements taken from inside the Kubernetes cluster itself. We measure the time it takes for the new container to start after it is removed as it is reported in the Kubernetes logs

For each MEC application, we carry out a set of 10 tests, covering both internal and external evaluations. In each of these tests, more than 10,000 samples were collected and analysed to ensure thorough evaluation and validation of our approach.



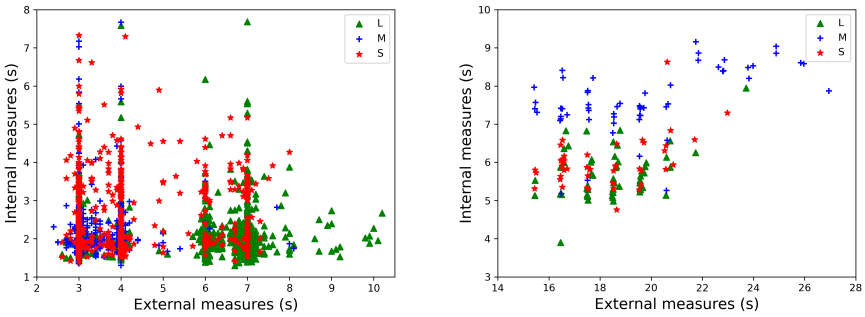
(a) MEC apps in containers (b) MEC apps in VMs

Fig. 4: External and internal relocation latencies for different MEC app sizes

In Figure 4 we can observe the delay boxplots of the relocation operation as perceived with the internal and external measurements for both VMs and containers hosting the chosen MEC applications. The results show that the relocation latency is not necessarily dependent on the application image size, which can be spotted for the S application in the case of Kubernetes, or the M applications in both the VM case. This leads to the conclusion that the image size has a smaller significance when compared to the complexity of the application itself, i.e. number of internal services that need to be configured and started. We have also computed the 95% confidence intervals that increase with the size of the edge service as the variance increases during the relocation process, which should be taken into account as an important parameter by the proactive MEAO.

A very interesting result is presented when comparing the external and the internal measurements in Figure 4. It seems that the time spent on relocation is quite short, in the range of 5s to 9s for VMs, and expectedly lower 1s to 3s for containers. However, the time reported with the external measurements is quite higher reaching 15s to 25s for VMs and 3s to 9s for containers. This means that there is a significant delay from the moment the NFVI decided the relocation has terminated to acknowledging this process by OSM and the HTTP server correspondingly. This discrepancy is of vital importance when aiming to build a proactive MEAO that is aware of the real-time status of relocations in progress. In addition, when comparing these values with the results obtained in [22] where the instantiation of MEC app in OpenNESS is reported to be in the range of 50s to 55s it can be concluded that the results obtained are several times lower leading to the conclusion that the implementation of a standalone Kubernetes cluster can provide much-improved performances when compared to the deployment inside a VIM solution.

Figure 5a represents the internal and external delay metrics of relocation are presented one against the other. Internal measurements without outliers show a range of 1s to 10s, while external measurements show 2s to 8s high variability. This variability re-exposes the importance of considering more than just the size



(a) Kubernetes containers relocation operations

(b) Microstack VMs live migration operations

Fig. 5: Internal vs external measures

regarding service performance for efficient service relocation management since there may be cases in which the S application takes a considerably longer time.

Figure 5b shows the combination of the obtained internal and external measurements and we can observe that internal values range from 3s up to 10s while external values go up to a range between 14s and 28s. It is obvious that the image sizes of these applications are considerably larger compared to the containers case and, hence, migration delays increase accordingly. We can observe that there is no clear location differentiation of the measurements corresponding to the three application sizes as in the case of the containers test. However, with these results, we can state that the overhead of OSM is considerably higher when considering applications deployed as VMs instead of as containers.

Performing these measurements is the first step to optimising the migration process of service hosting containers in complex environments such as MEC-NFV. It is essential that one primarily analyses the obtained data and understands the dependencies and relationships that are present in the collected dataset. The next step is to use the gathered data as one of the inputs for training proactive algorithms that improve the orchestrator's resource management during migration, making better decisions, reducing latency and improving real-time resource management. The main idea is to use the latency data, along with other contextual features such as container size, application type, network conditions, and resource availability, as input features for an AI-based model. When training the model historical latency data can be used to capture patterns and trends that help predict future migration needs. In addition, the data can be labelled with optimal migration times or decisions based on past successful migrations with minimal disruption. Labelling can also be performed for instances where resource allocation is suboptimal, thus helping the algorithm learn to predict and avoid such situations. In this way, an algorithm can be developed so that it learns to predict the best times to migrate containers based on the current and predicted future state of the network and resources, minimising latency

and optimising resource use. Finally, the model can be integrated with the real-time monitoring environment to make proactive decisions. The model can also be continuously updated with new latency measurements, which will allow it to adapt and improve over time.

6 Discussion and Conclusions

The commitment to join 5G and MEC technologies based on NFV seems to be a winner in solving the requirements of applications in real-time. With containers gaining prominence in application virtualisation with ultra-low latency requirements, in this work, we analysed the performance of the procedure of relocation of containers in an OSM-based MEC-NFV system with a standalone cluster versus the traditional OSM setup that uses virtual machines.

While OSM has continued to be improved in its new releases, the implementation of standalone Kubernetes cluster for MEC applications based on KNFs lags behind especially when it comes to seamless support for container relocation. Our performance results and comparison however show that the performances of standalone containers are significantly improved when compared to VMs, but also inside a VM container implementations. With the research activities being focused on the development of intelligent MEAOs with proactive capabilities for optimal placement and relocation of cloud-native container-based MEC applications, the ability to perform these functionalities and supply the MEAO with real-time status and monitoring updates becomes crucial.

Adapting to online services changing demands and improving the end-user experience underscores this importance in modern network architectures. To address the challenges while designing for maximum gain we will continue our efforts towards the development of full support for CNFs in an ETSI MEC NFV-compliant architecture with the aim to support the implementation of proactive service relocation.

References

1. Hani Attar, Haitham Issa, Jafar Ababneh, Mahdi Abbasi, Ahmed AA Solyman, Mohammad Khosravi, Ramy Said Agieb, et al. 5g system overview for ongoing smart applications: Structure, requirements, and specifications. *Computational Intelligence and Neuroscience*, 2022:1–11, 2022.
2. Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167:106984, 2020.
3. ETSI Gs MEC. V2. 1.1. multi-access edge computing (mec); framework and reference architecture, 2022.
4. Mobile Edge Computing. Deployment of mobile edge computing in an nfv environment. *ETSI Group Report MEC*, 17:V1, 2018.
5. Network Functions Virtualisation ETSI. Network functions virtualisation (nfv). *Management and Orchestration*, 1:V1, 2014.

6. Tao Ouyang, Zhi Zhou, and Xu Chen. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications*, 36(10):2333–2345, 2018.
7. Sergej Svorobej, Malika Bendechache, Frank Griesinger, and Jörg Domaschka. Orchestration from the cloud to the edge. *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing*, 4:61–77, 2020.
8. Mohammad Faraji-Mehmandar, Sam Jabbehdari, and Hamid Haj Seyyed Javadi. A self-learning approach for proactive resource and service provisioning in fog environment. *The Journal of Supercomputing*, 78(15):16997–17026, 2022.
9. Cristina Bernad, Vojdan Kjorveziroski, Pedro Juan Roig, Salvador Alcaraz, Katja Gilly, and Sonja Filiposka. Multi-access edge computing smart relocation approach from an nfv perspective. In *International Conference on ICT Innovations*, pages 38–48, Berlin, DE, 2022. Springer.
10. Marie Siew, Kun Guo, Desmond Cai, Lingxiang Li, and Tony QS Quek. Let's share vms: Optimal placement and pricing across base stations in mec systems. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10, New York, NY, 2021. IEEE.
11. Lei Zhao and Jiajia Liu. Optimal placement of virtual machines for supporting multiple applications in mobile edge networks. *IEEE Transactions on Vehicular Technology*, 67(7):6533–6545, 2018.
12. Ali Asghari, Hossein Azgomi, Ali Abbas Zoraghchian, and Abbas Barzegarinezhad. Energy-aware server placement in mobile edge computing using trees social relations optimization algorithm. *The Journal of Supercomputing*, 1(1):1–29, 2023.
13. The-Vi Nguyen, Nhu-Ngoc Dao, Wonjong Noh, Sungrae Cho, et al. User-aware and flexible proactive caching using lstm and ensemble learning in iot-mec networks. *IEEE Internet of Things Journal*, 9(5):3251–3269, 2021.
14. Anahita Mazloomi, Hani Sami, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. Reinforcement learning framework for server placement and workload allocation in multiaccess edge computing. *IEEE Internet of Things Journal*, 10(2):1376–1390, 2022.
15. Xiaohan Jiang, Peng Hou, Hongbin Zhu, Bo Li, Zongshan Wang, and Hongwei Ding. Dynamic and intelligent edge server placement based on deep reinforcement learning in mobile edge computing. *Ad Hoc Networks*, 145:103172, 2023.
16. Azzedine Boukerche et al. *A Comparative Study on Service Migration for Mobile Edge Computing Based on Deep Learning.* PhD thesis, Université d'Ottawa/University of Ottawa, 2023.
17. Vinicius Ferreira, João Bastos, André Martins, Paulo J. Araújo, Nicolás Lori, João Faria, António Costa, and Helena Fernández López. Netedge mep: A cnf-based multi-access edge computing platform. In *2023 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, New York, NY, 2023. IEEE.
18. Balázs Sonkoly, Dávid Haja, Balázs Németh, Márk Szalay, János Czentye, Róbert Szabó, Rehmat Ullah, Byung-Seo Kim, and László Toka. Scalable edge cloud platforms for iot services. *Journal of Network and Computer Applications*, 170:102785, 2020.
19. Adrián Pino, Pouria Khodashenas, Xavier Hesselbach, Estefanía Coronado, and Shuaib Siddiqui. Validation and benchmarking of cnfs in osm for pure cloud native applications in 5g and beyond. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, New York, NY, 2021. IEEE.
20. ETSI. Open source mano (osm) project, December 2023.

21. Girma M Yilma, Zarrar F Yousaf, Vincenzo Sciancalepore, and Xavier Costa-Perez. Benchmarking open source nfv mano systems: Osm and onap. *Computer communications*, 161:86–98, 2020.
22. Pablo Fondo-Ferreiro, Alberto Estévez-Caldas, Rubén Pérez-Vaz, Felipe Gil-Castiñeira, Francisco Javier González-Castaño, Santiago Rodríguez-García, Xosé Ramón Sousa-Vázquez, Diego López, and Carmen Guerrero. Seamless multi-access edge computing application handover experiments. In *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, New York, NY, 2021. IEEE.
23. Mohammed A Hathibelagal, Rosario G Garroppo, and Gianfranco Nencioni. Experimental comparison of migration strategies for mec-assisted 5g-v2x applications. *Computer Communications*, 197:1–11, 2023.
24. Sagar Arora, Adlen Ksentini, and Christian Bonnet. Cloud native lightweight slice orchestration (cliso) framework. *Computer Communications*, 213:1–12, 2024.
25. Karamjeet Kaur, Veenu Mangat, and Krishan Kumar. A review on virtualized infrastructure managers with management and orchestration features in nfv architecture. *Computer Networks*, 217:109281, 2022.
26. ETSI. Using kubernetes-based vnfs (knfs), December 2023.
27. Cristina Bernad Canto, Pedro Juan Roig, Sonja Filiposka, Salvador Alcaráz Carrasco, and Katja Gilly. Challenges of implementing nfv-based multi-access edge computing environments. In *2021 29th Telecommunications Forum (TELFOR)*, pages 1–4, New York, NY, 2021. IEEE.
28. Theodoros Tsourdinis, Nikos Makris, Serge Fdida, and Thanasis Korakis. Drl-based service migration for mec cloud-native 5g and beyond networks. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pages 62–70, 2023.
29. Vojdan Kjorveziroski and Sonja Filiposka. Kubernetes distributions for the edge: Serverless performance evaluation. *The Journal of Supercomputing*, 78(11):13728–13755, July 2022.