Angel Cañete, Mercedes Amor, Lidia Fuentes,

HADES: An NFV solution for energy-efficient placement and resource allocation in heterogeneous infrastructures,

Journal of Network and Computer Applications,

Volume 221,

2024,

103764,

ISSN 1084-8045,

https://doi.org/10.1016/j.jnca.2023.103764.

(https://www.sciencedirect.com/science/article/pii/S1084804523001832)

Abstract: Network Function Virtualization (NFV) aims to replace traditional network functions running in proprietary hardware with software instances (i.e., Virtual Network Functions, VNFs) embedded in general-purpose virtualization solutions. Aware that the transition to a fully virtualized network infrastructure will pay a high energy cost, especially in IoT systems composed of a myriad of devices, energy efficiency is one of the key innovative targets of future networks. Edge computing should be considered in IoT environments to save time and energy by processing data near the producer devices. However, applying NFV in the context of IoT/Edge/Cloud environments complicates the placement of VNFs, due to the inherent heterogeneous nature of such environments and the variety of resource demands. This paper proposes an energy-aware placement of service function chains of VNFs and a resource-allocation solution for heterogeneous edge infrastructures that considers the computation and communication delays according to the VNFs' location in the infrastructure. The solution has been integrated with the ETSI-sponsored project Open Source Management and Orchestration (OSM) as an extension called HADES, which allows the configuration of VNFs and their subsequent resource allocation and deployment at the edge, minimizing energy consumption and ensuring a quality of service. We have applied the deployment of augmented reality services in real and simulated scenarios. The results show up to a 59% reduction in power consumption and QoS compliance in all scenarios considered compared to default OSM placement and four other allocation policies. We prove that our solution has negligible power overhead, and validate the scalability and applicability of HADES.

Keywords: Energy efficiency; B5G; VNF placement; Edge computing; Open source MANO; IoT; Feature Models

HADES: An NFV solution for energy-efficient placement and resource allocation in heterogeneous infrastructures

Angel Cañete[a,*], Mercedes Amor[a], Lidia Fuentes[a]

[a]*CAOSD Group, Lenguajes y Ciencias de la Computación, Bulevar Louis Pasteur, 35. Campus de Teatinos, Málaga, 29071, Spain*

Abstract

Network Function Virtualization (NFV) aims to replace traditional network functions running in proprietary hardware with software instances (i.e., Virtual Network Functions, VNFs) embedded in general-purpose virtualization solutions. Aware that the transition to a fully virtualized network infrastructure will pay a high energy cost, especially in IoT systems composed of a myriad of devices, energy efficiency is one of the key innovative targets of future networks. Edge computing should be considered in IoT environments to save time and energy by processing data near the producer devices. However, applying NFV in the context of IoT/Edge/Cloud environments complicates the placement of VNFs, due to the inherent heterogeneous nature of such environments and the variety of resource demands. This paper proposes an energy-aware placement of service function chains of VNFs and a resource-allocation solution for heterogeneous edge infrastructures that considers the computation and communication delays according to the VNFs' location in the infrastructure. The solution has been integrated with the ETSI-sponsored project Open Source Management and Orchestration (OSM) as an extension called HADES, which allows the configuration of VNFs and their subsequent resource allocation and deployment at the edge, minimizing energy consumption and ensuring a quality of service. We have applied the deployment of augmented reality services in real and simulated scenarios. The results show up to a 59% reduction in power consumption and QoS compliance in all scenarios considered compared to default OSM placement

*Corresponding author

Email addresses: angelcv@uma.es (Angel Cañete), map@uma.es (Mercedes Amor), lfuentes@uma.es (Lidia Fuentes)

and four other allocation policies. We prove that our solution has negligible power overhead, and validate the scalability and applicability of HADES.

Keywords: energy efficiency, B5G, VNF placement, Edge Computing, Open Source MANO, IoT, Feature Models

---

1. Introduction

In today's world, every process and activity is turning more sustainable, mainly to mitigate global warming and alleviate the current crisis of energy production. In this context, main IT (Information Technology) providers (e.g., Google or Microsoft), as well as network operators and telecommunication companies are promoting greener solutions in terms of both, reduction of energy consumption (EC), and prioritizing the use of infrastructure fully supported by renewable energy. In fact, incoming generations of mobile networks (i.e., B5G/6G) are targeting the objective of energy efficiency, while promising more capacity, lower latency, and reliability. The capabilities of 5G/B5G technology enable a variety of connected devices to communicate faster, driving the growth and commercial success of IoT (Internet of Things) systems. However, IoT services usually feed the network with an enormous volume of data usually processed in the Cloud, which negatively contributes to the goal of energy efficiency [1]. In this sense, the combined use of Edge and Cloud infrastructures would be a promising solution to reduce the energy footprint of IoT applications, since some computation is performed in the frontier of the network [2], i.e., next to the data producer devices.

Over the past decade, network technologies have evolved to offer a set of enabling technologies that leverage the advantages of IoT/Edge/Cloud environments [3], with the promise of supporting energy-efficient IT solutions. Network Function Virtualisation (NFV) is one of the key technologies that facilitate power management within multiple independent virtual networks, called network slices, that share a physical infrastructure [1]. NFV main contribution is the virtualization of network functions, such as firewalls, multimedia streaming, or proxies, and embedding their execution in virtual machines (VMs) or containers. These virtualized functions, so-called Virtual Network Functions (VNFs) can run in any computing device (i.e., mobile phone, edge device, cloudlet, etc.), and can be bound together into Service Function Chains (SFCs) of connected (virtual) network services. NFV and software-defined networks [1] allow to manage resources and SFCs/VNFs

within a network slice, independently and individually to fulfil the wide variety of demands requested by different web services, application domains, customers, or operators. The problem of deploying SFCs/VNFs inside a virtual network slice, satisfying a set of demands and device constraints is known as VNF placement, and it is an important area of recent research [4]. In addition, being conscious that the transition to a fully virtualized network infrastructure will pay a high energy cost [5], energy efficiency is one of the key innovative targets of future communication systems.

However, existing approaches to VNF placement present several weaknesses that hamper their applicability and usefulness. Leveraging edge computing presents challenges. IoT developers must decompose application functions into tasks and optimally allocate them to edge nodes. This task identification is typically manual in existing edge computing methods. Most current approaches fail to consider the specific configuration demands and particularities of VNFs when creating generalized deployment descriptors. This lack of consideration undermines the accuracy and effectiveness of deployment. Additionally, some hardware dependencies of VNFs and their impact on QoS (Quality of Service) requirements are neglected, resulting in suboptimal placement decisions that do not meet customer demands. Furthermore, many existing solutions overlook the importance of minimizing energy consumption in VNF chains, focusing solely on optimizing energy rates as an objective and non-considering the type of energy (renewable or not) the nodes are powered with. Finally, the majority of solutions are proprietary, making it challenging to reuse them in different network environments like B5G.

In order to address some of the drawbacks of these works, this paper addresses the VNF placement in heterogeneous IoT/Edge/Cloud environments focusing on minimizing energy consumption while assuring QoS, following the N-MAPE-K loop, an extension of the classical Monitor-Analyze-Plan-Execute loop over a shared Knowledge (MAPE-K [6]) adapted for Networks, defined in the DAEMON project [7]. The proposed solution facilitates the adoption of edge computing through the utilization of variability models, a technique prevalent in Software Product Lines (SPL) and proven to be useful for this purpose [8, 9]. These models depict the breakdown of an application's functions into services and tasks, including the amount of resources they require for their execution. During the deployment step, it considers requirements that are not commonly addressed in current SFC/VNF placement decisions such as disk, power feeding, or peripherals, and distinguishes nodes powered by renewable energy as (partially or fully) non-consumers, strategically deploying VNFs

in energy-saving nodes and promoting eco-friendly practices. The solution has been integrated as an extension of the Open Source Management and Orchestration (OSM) project [10], an ETSI (European Telecommunications Standards Institute) sponsored project, and tested in real scenarios. OSM provides a VNF manager, an NFV orchestrator and supports connection to third-party Virtualized Infrastructure Managers (VIMs), e.g., OpenStack, OpenVIM, VMWare. In addition, OSM supports container technology (via Kubernetes [11]) to enable the design and implementation of Kubernetes-based Network Functions (KNFs) [10] running inside containers. Our solution, named HADES, extends OSM by enabling the placement and configuration of VNFs/KNFs and their subsequent resource allocation and deployment at the edge, minimizing energy consumption and latency. Furthermore, HADES enables deployments that take into account multiple VIMs, a feature not natively supported by OSM. Furthermore, the use of Kubernetes endows the infrastructure with fault tolerance and high availability.

HADES placement solution is evaluated using augmented reality (AR) services (as KNFs in an SFC) in different scenarios. Using a physical energy meter, the energy consumption of the HADES placement solution is compared with the default OSM placement. Furthermore, the quality of service (QoS) obtained with both deployments is tested. The results show up to a 51% reduction in EC and QoS compliance in all (physical) scenarios considered. These experiments have been extended using a simulated environment, obtaining up to 59% reduction in EC compared to other five allocation policies. The energy overhead of the solution has been evaluated, being negligible when compared to the reduction in EC obtained. Its scalability has also been evaluated, being proved that is fast enough to be applicable in real environments (even when running on commercial computers). The promising results can help encourage operators to include energy awareness and promote the use of KNFs and take full advantage of containerization in NFV.

The remainder of this document is organized as follows. Section 2 introduces the NVF Management and Orchestration platforms, and presents and discusses the related works. Section 3 presents HADES and its integration with OSM, while Section 4 details the different modules that make up our proposal. Section 5 evaluates our proposal by applying it to the deployment of an augmented reality application on a physical edge infrastructure, evaluates the reduction in energy consumption obtained in both physical and emulated scenarios, and evaluates its scalability and energy overhead. Section 6 discusses the threats to validity, while Section 7 provides background information.

Finally, Section 8 concludes the paper and presents future work.

2. Related work

This section addresses the state of the art in recent existing approaches for SFC placement, showing that is still an open issue and motivates our targeted research effort on SFC placement.

2.1. SFC placement

In NFV, a requested service is implemented by a set of VNFs that are connected by data flows in a defined order. SFC [12] allows defining an ordered sequence of network functions (NFs) to provide end-to-end services [13]. Typically, a network SFC provides data stream services where data pass through a sequence of functions, which also may be shared between applications (e.g., a video stream passing through the rendering, tracking, firewall and encryption services of software-defined wide area networks). In NFV, the SFC can be set by placing the constituent VNFs in a very efficient and agile way. Traditionally, SFC placement is usually addressed as a code or task offloading [14, 15], also described as a task assignment problem. In general terms, task assignment approaches differ in the objective(s) that drive the optimization (e.g., reducing the energy consumption, latency, etc.), the nodes involved (user node, all nodes, etc.), granularity level, the task model and mechanisms used to obtain the solution, and the type of infrastructures and/or nodes considered [16, 17, 18]. However, the placement of SFC functions is a somewhat complex problem, as the solution must not only satisfy the requirements and demands of each NF but also fulfil a number of constraints and requirements derived and imposed from the temporal order and/or logical dependencies between the NFs.

In [19], authors address the VNF placement and resource optimization problem in NFV and edge computing-enabled networks focusing on a set of known SFC requests. Their goal is to minimize the total resource consumption by formulating the VNF placement problem as an Integer Linear Programming (ILP) model that considers the CPU (Central Processing Unit), memory, and bandwidth of nodes and VNFs. The authors also propose a heuristic solution to resolve the SFC placement problem, whose performance is evaluated in the paper. The work in [20] presents a hierarchical architectural framework to address the SFC placement, adhering to the general principles outlined by major standardization organizations. The proposed solution uses ILP and four

heuristic algorithms and considers the CPU and memory of the nodes. The approach entails periodic resetting of the routing flows and VNF allocation. The link status is periodically monitored to determine if the recalculation of the routing path is required. Hence, every time a new routing recalculation is needed the placement algorithms take into consideration the latest status update of the resources, which can derive to a VNF reallocation. Huang et al. [21] propose a binary cloud-MEC (Mobile Edge Computing) collaborative task-offloading model to minimize latency and energy consumption. Their solution involves offloading resource-demanding or delay-tolerant tasks to a single cloud server, and delay-sensitive tasks to MEC servers, resulting in an energy consumption reduction of up to 13.73%. Yang et al. [22] present an approximately lossless model compression mechanism to ease the computing burden of virtual network embedding over multilayer elastic optical networks for edge-cloud collaborative services. First, they establish an integer quadratic constraint programming model for the problem, and then they apply model compression based on virtual link mapping cost Estimation to shield the variables and constraints. Finally, they solve the resulting model using Hopfield neural networks and evaluate its performance using a benchmark. The proposed solution reduces the blocking ratio by about 35% in the single layer and by about 50% in the multilayer elastic optical networks scenario. The model manages nodes' resources and the requirements of the computing services in a single variable, making it impossible to manage each type of resource separately. Authors in [23] present a solution for the resource optimization and delay-guarantee VNF placement problem in cloud networks. To this aim, authors consider CPU, memory and bandwidth resources, placing VNFs in heterogeneous servers using a partial offloading scheme. The problem is solved using a two-phase optimization solution, consisting of a mapping phase that maps SFC requests with servers, and an adjustment phase that optimizes the placement of VNFs with VNF requests. Bunyakitanon et al. [24] present Auto-3P, an Autonomous module for Virtual Network Functions Performance Prediction and Placement at network cloud and edge facilities. Auto-3P enhances the autonomous placement abilities of MANOs (Management and Orchestration) by taking into account the availability of resources on hosting nodes as well as the potential impact of a VNF node's placement decision on the overall end-to-end performance of the service. The proposed model considers CPU, memory, and bandwidth of nodes and addresses the VNFs placement with machine learning algorithms (k-nearest neighbours regression, decision tree, and support vector regression).

In [25] authors propose machine learning models, in particular neural networks, that can perform auto-scaling by predicting the required number of VNF instances based on the traffic demand (VNF scaling), using the radio access traffic traces collected over a real-operator commercial network. Then, they employ ILP to solve the SFC placement problem, where each SFC represents a service requested by a user with end-to-end latency and data in the radio access. The model considers CPU, memory, disk and bandwidth of nodes and VNFs to carry out the placement. Additionally, the authors present a proof-of-concept implementation of an NFV management and orchestration framework for enterprise WLANs. The work in [26] proposes a supervised machine learning model that configures different weights for various entities, based on differentiated weight graph convolutional neural networks, to predict optimal VNF placements and enhance performance. The model integrates probabilistic outputs of the machine learning model with a maximal weight matching heuristic and expert knowledge-based probability reshaping to ensure VNF placements are feasible. The proposed model considers the CPU, GPU (Graphics Processing Unit), memory, and storage requirements of the VNFs and node resource capacities. The authors also evaluate the time efficiency of the proposed methods. In [27] the authors propose a solution to address the VNF placement with the aim of satisfying the delay constraints and reducing the consumption of network resources. The SFC placement is solved as a Mixed Integer Non-linear Programming (MILNP) and a heuristic algorithm is proposed based on the relaxation and delay check. With the aim of minimizing energy consumption, in [28] authors formulate the VNF placement problem as a decision tree search. The parameters considered during the process are the CPU nodes, their type, and the bandwidth of the links. They consider that the amount of CPU required by the VNFs is fixed, which weakens the applicability of the proposal in real heterogeneous infrastructures. With the same objective, authors in [29] propose an NFV scheduling heuristic algorithm to map and schedule traffic flows, reducing the EC in the entire infrastructure up to 46%. In [30] authors propose a resource and energy-aware strategy for the placement of SFC of IoT applications in edge–cloud environments. They apply a heuristic SFC placement strategy to maximize the number of accepted chains by balancing the resource usage between edge and cloud tiers. Then the power consumption of the physical equipment can be minimized by optimizing VNF placement at both cloud and edge. However, this is an NFV ad-hoc solution, which can not be easily integrated with any of the existing standard-aligned NFV frameworks and

it is not clear how the placement algorithm would work in heterogeneous infrastructures. Authors in [31] employ non-linear integer programming and a modified Viterbi algorithm that relies on multi-stage graph modelling and a modified Dijkstra's algorithm to tackle the SFC placement in cloud infrastructures. However, their model solely considers the CPU of nodes, using absolute values (such as 1 CPU core), which confines its use in heterogeneous infrastructures found in the real world. The work in [32] introduces an ILP model and a heuristic algorithm that extends node aggregation to develop a function graph at the domain level with a reduced size. However, the model only accounts for bandwidth, and node resources are considered in absolute terms (expressed by a single variable and not differentiating between resource types), which restricts its applicability in real-world scenarios.

Table 1 shows a comparison of how the considered approaches address the placement of NFs in an SFC problem. Concretely, Table 1 compares these approaches in terms of: (i) the objective or objectives that drive the placement optimization of SFC (such as energy consumption, latency, performance); (ii) the third column indicates the type of infrastructure considered for placement (edge, cloud or both); (iii) the fourth column enumerates the requirements and constraints of the placement decision (such as the type of targeted nodes, the available RAM (Random-Access Memory), storage, or specific a hardware/server configuration); (iv) next column shows the technique or mechanism(s) used by the placement algorithm; (v) the sixth column shows if the approach allows flexible resource allocation, i.e., whether the CPU requirements are fixed in the model for each VNF of the SFC, or whether the solution allows dynamic CPU allocations to meet end-to-end latency requirements; (vi) the seventh column indicates if the approach performs or supports real-time monitoring of current resources and if it is able to relocate already deployed VNFs when it is beneficial to the infrastructure; (vii) the eighth column specifies whether the proposal allows to adapt the SFC' VNFs special needs to the available infrastructure and resources (e.g., CPU to be assigned to each VNF); and, (viii) finally, the last column indicates whether the approach is integrated with a widely used solution to facilitate adoption, or whether it is proprietary.

2.2. Current works limitations

In [33] authors analyze the SFC placement requirements and limitations of current works. They recommend some useful assumptions that need to be considered as part of the VNF placement problem. First, VNFs are

Table 1: Related works on SFC Placement

Work	Objective(s)[a]	Target infrastructure	Resources considered[b]	Solution mechanism(s)[c]	Flexible resource allocation	VNF reallocation	VNF configuration	Kind of solution
[19]	RO	Edge	CPU, M, B	ILP, HA	×	×	×	Proprietary
[20]	P	Cloud	CPU, M	ILP, HA	×	✓	×	Proprietary
[21]	L, EC	Edge, Cloud	Not specified	IA	×	×	×	Proprietary
[22]	VNRC, BR	Edge, Cloud	CPU, GPU, M, D	IQCP, VLMCE, HNN	×	×	×	Proprietary
[23]	RO	Cloud	CPU, M, B	TPO	×	×	×	Proprietary
[24]	Pe	Edge, Cloud	CPU, M, B	ML	×	×	×	OSM extension
[25]	Pe	Edge	CPU, M, D, B	ILP, HA	×	×	✓	NFV Mano
[26]	Pe	Edge	CPU, GPU, M, D	ML	×	×	×	Proprietary
[27]	RO	Edge	CPU, B	MILNP, HA	×	×	×	Proprietary
[28]	EC	Cloud	CPU, T, B	DTS	×	×	×	Proprietary
[29]	EC	Cloud	CPU	HA	×	×	×	Proprietary
[30]	EC	Edge, Cloud	CPU, M, B	HA	×	×	×	Proprietary
[31]	EC	Edge	CPU (cores), B	NLIP, MVA	×	×	×	Proprietary
[32]	EC	Cloud	B, AbR	ILP, HA	×	×	×	Proprietary
Our approach	**EC**	**Edge, Cloud**	**CPU, GPU, M, D, B, T, Per**	**MILP**	✓	✓	✓	**OSM extension**

[a] BR: Blocking ratio; EC: Energy consumption; L: Latency; Pe: Performance; RO: Resource optimization; VNRC: VNR cost. [b] AbR: Absolute resources; B: Bandwidth; D: Disk; M: Memory; Per: Peripherals; T: Node type. [c] DTS: Decision-tree search; IA: Iterative algorithm; HA: Heuristic Algorithm; HNN: Hopfield neural network; ILP: Integer Linear Programming; NLIP: Non-linear Integer Programming; MILNP: Mixed Integer Non-linear Programming; ML: Machine learning; MVA: Modified Viterbi algorithm; TPO: Two-phase optimization; ICQP: Integer Quadratic Constraint Programming; VLMCE: Model compression based on Virtual Link Mapping Cost Estimation.

characterized by their own configuration and specificities, thus, creating generalized deployment descriptors is a challenge. Indeed, describing specific configuration demands of VNFs as part of their descriptor is not usually considered by current solutions. Analyzing the works of Table 1, none of them address appropriately this requirement. We address this requirement (fourth column) by considering VNFs' particular specificities as part of the problem model. Concretely, our approach considers that some VNFs may require to be executed in a node with a certain hardware configuration, such as having a GPU optimized for deep learning, a certain CPU configuration, etc. In addition, we consider that some VNFs running in the user plane usually demand special peripherals, such as video cameras or live streaming equipment. Regarding the resources available in the nodes, our approach includes all those considered by other approaches and additionally the peripheral ones (fourth column). This involves not only checking whether the nodes have certain peripherals but also checking that the characteristics of these peripherals ensure the quality of service desired by the user (e.g., the resolution of a camera).

Second, the implementation environment of each node of the network affects the list of VNFs that can be deployed in a particular network slice. Therefore, VNF chaining should take into account the hardware dependencies of those VNFs as part of the placement solution to ensure certain customers' QoS demands. This requirement also allows, for example, service providers to deploy certain VNFs, such as VoIP-related services, in high-capacity nodes to meet a desired QoS. After reviewing the papers of Table 1 we found that none

consider this requirement as part of the problem description (sixth column). Indeed, none of the previous works (except ours): (i) allows flexible resource allocation (according to the existing resources and ensuring a minimum QoS); and (ii) supports the configuration or reallocation of VNFs as part of the solution (only [20, 25] allow VNF reallocation or VNF configuration, as mentioned before).

Another important requirement is minimizing the energy consumption of the VNF chains [34] (second column), which is not always considered in this kind of work. Although many energy-aware SFC placement approaches have been proposed ([21, 28, 29, 30, 31, 32]), they consider the energy rate only as an objective to optimize. However, what we should be looking for is a sustainable VNF placement solution, which means, among other things, distinguishing those nodes that use clean energy from others. Our approach addresses this requirement by considering nodes powered by renewable energy as (partially or fully) non-consumers. Thus, as part of our energy reduction strategy, we also prioritize the deployment of VNFs in those nodes where energy savings are higher.

Regarding the technique used to perform SFC placement, there is a high diversity of algorithms. A large number of works use variations of ILP, and others use heuristics or machine learning algorithms (fifth column of Table 1). To avoid the scalability problems of traditional optimization approaches, machine learning approaches are preferred. Machine learning approaches may have the advantage of not requiring a complete or in some cases previous knowledge of the system, and can quickly provide a placement decision, being reinforcement learning, deep learning and deep reinforcement learning widely adopted techniques. However, some papers surveying works [35, 36] that address VNF placement in edge and cloud computing [35, 36] conclude that machine learning approaches have still important limitations such as: (i) the simulation and learning settings consist of a homogeneous set of VNFs requirements and nodes; (ii) they only consider energy consumption as one of the objectives to optimize during placement decisions; (iii) they can not guarantee to find an optimal solution, which affect solution accuracy [36]; (iv) in supervised learning approaches the time and energy cost of training does not outweigh its advantages. On the contrary, in our approach, we have tested the energy cost of our algorithm finding that is negligible.

Finally, except [24] and [25] all the analyzed solutions are proprietary, being difficult to reuse in different network environments like B5G. Our main conclusion is that our HADES approach provides a useful and distinct solution,

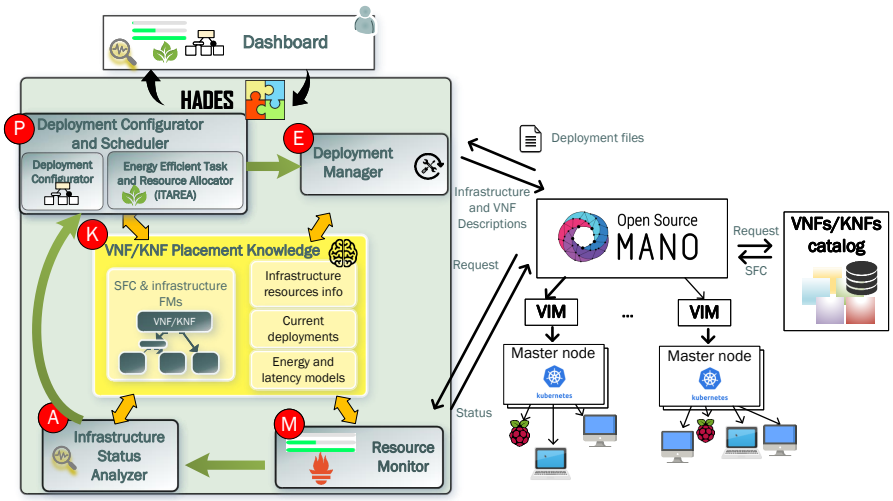

Figure 1: General overview of our approach

addressing important requirements raised in the literature and providing a modular solution implemented on the OSM standard platform.

3. HADES Overview

This section presents HADES, whose main contribution is to extend OSM's VNF/KNF placement by minimizing energy consumption and ensuring the QoS of supporting services. Additionally, it supports VNF/KNF deployment through OSM by enabling zero and one-touch management operations. Our energy-aware VNF placement solution manages heterogeneous edge infrastructures and considers the energy consumption and the computation and communication delay within an NFV infrastructure. HADES also considers requirements that are not commonly addressed in the VNF placement, such as memory, storage, or a specific hardware/server configuration (e.g., a GPU, a specific network card, etc.).

Figure 1 depicts the architecture of our approach showing the integration of our platform HADES with the Open Source MANO [10] framework. The HADES architecture follows the N-MAPE-K loop, an extension of the classical Monitor-Analyze-Plan-Execute loop over a shared Knowledge[6]) adapted for Networks[7]. MAPE-K is one of the most influential reference control models for autonomous and self-adaptive systems. We will describe the HADES components implementing the N-MAPE-K loop:

- ***M****onitor*: The *Resource Monitor* module (M in Figure 1) monitors the resource usage of the nodes (CPU, memory, disk, and links' bandwidth) in real-time using Prometheus and Kubernetes monitoring tools. This is the first step of the N-MAPE-K loop, the one in charge of collecting the information that feeds the loop. This information is persistently stored as a time series in the K component, which is the central part of the loop storing all the information needed by each N-MAPE-K component.

- ***A****nalyze*: This is the second step of the loop. The output of M, that is, the current status of the infrastructure is analyzed (in component *Infrastructure Status Analyzer* labelled as A) to check the availability of the infrastructure's capabilities and check if there is a more energy-efficient allocation of existing deployments (taking into account the cost of migration).

- ***P****lan*: Once, the A function decides a re-deployment of the VNFs should be performed, the P task is invoked to generate a new optimal solution adapted to the current network status. The *Deployment Configurator and Scheduler* (labeled as P) contains the *energy effIcient TAsk and REsource Allocator* (iTAREA), which is responsible for computing the optimal (energy-aware) placement of VNF/KNF by executing the iTAREA algorithm. The information about the amount of resources (CPU, RAM, disk, and bandwidth), the resource requirements of applications, and current characteristics of the infrastructure (e.g., the list of nodes powered by renewal energy) are taken from the central knowledge K module, keeping this information persistently in a relational database. We use a formal variability model called Feature Model (FM [37]) to specify IoT infrastructures and generate valid configurations (see the Deployment Configurator component), and also to specify application features and their demands [17]. Also, the SFC is modelled with a graph that specifies the sequence of VNFs and data flow.

- ***E****xecution*: Finally, this is the last step of the loop. The VNF/KNF *Deployment Manager* module (component E) executes the deployment (placement and resource allocation) computed in the previous step. To do so, it uses the OSM's North Bound Interface API to send the deployment file containing the KNF/VNF new locations to OSM. OSM's North Bound Interface is RESTful and follows the ETSI SOL005 standard.

- ***K**nowledge*: The central *VNF/KNF Placement Knowledge* component (tagged with a K) maintains the information retrieved from KNF/VNF repository, the KNF/VNF configuration models, the energy and latency models, monitored infrastructure's status, and software and hardware infrastructure characteristics (CPU power, peripherals, memory, etc.).

These components conform the HADES workflows of the different activities HADES performs. In addition, HADES includes a Dashboard web-based component, which provides a graphical interface so that the user can control, monitor and consult the activities of HADES and the data and state of the components of the N-MAPE-K loop (see Figure 1).

Specifically, HADES extends and enhances OSM capabilities in the following ways:

- *Minimization of energy consumption:* HADES minimizes the energy consumption of the deployments through an energy-aware placement decision.

- *Assisted generation of application configuration:* HADES allows the selection of both application and QoS-related features prior to deployment, avoiding the configuration of non-supported applications.

- *QoS assurance:* The selected functional and non-functional characteristics of the configured application are considered during the deployment phase.

- *Optimization of resource allocation:* HADES establishes a minimum and maximum range of resource usage (e.g., CPU) that ensures QoS while minimizing resource allocation.

- *Deal with infrastructures composed of multiple VIMs:* Natively, OSM deploys all the KNFs belonging to the same application in the same VIM. However, in infrastructures with more than one VIM, it is desirable to consider the available resources to select the most appropriate. HADES allows the deployment of KNFs belonging to the same service or application considering all the available VIMs, making better use of the available infrastructure resources.

4. HADES operation over OSM

This section presents how the HADES modules operate and interact with the OSM components for the optimal placement and deployment of VNFs/KNFs in a heterogeneous NFV infrastructure (see Figure 1). As mentioned before, OSM can work with Kubernetes (K8s). There are two modes to represent a K8s cluster in OSM: inside a VIM (single-net and multi-net), where OSM will deploy KNFs and manage containers through a VIM; or outside a VIM, allowing to interact directly with K8s and network services, which are instantiated through containerized KNFs. HADES adheres to the first mode because it allows the use of emulated and real VIMs without changing the KNFs. Moreover, the first approach is the one recommended by OSM, since KNFs are part of a more complex network context of a carrier network organized in terms of VIM and enables OSM to monitor the workloads running in this K8s cluster[10].

4.1. Resource Monitor

The *Resource Monitor* is responsible for tracking the status and resource usage of the infrastructure. Its main functions are to check the number of running VIMs, their characteristics and their associated cluster (if any); and to obtain real-time information about the workload of the CPU, memory, disk, and data transmitted. On the other hand, this module obtains energy consumption metrics about the running services.

To obtain information about the infrastructure's hardware in use, the *Resource Monitor* module uses Prometheus through OSM. Prometheus is a widely adopted open-source monitoring system for services and applications that run in containers and allows for gathering these metrics, such as CPU usage, bandwidth usage, and RAM usage. It has a multidimensional data model and a flexible query language and integrates aspects ranging from client-side instrumentation to alerts. Prometheus is installed in the K8 cluster and registers K8s cluster information in OSM [10]. HADES collects this data from OSM and stores it in the *VNF/KNF Placement Knowledge*, which maintains this information persistently in a database as time series data.

For the VNF/KNF placement, HADES internally estimates the energy consumption using an energy model (explained later) and monitors the energy consumption in real-time using Prometheus and Kubernetes. However, HADES can make use of different energy models and monitoring tools without affecting the deployment algorithm. What is important is not what is the

exact energy consumption value, but that the energy model estimates the energy footprint of KNFs in different devices, and under variable traffic in a coherent way.

4.2. Infrastructure Status Analyzer

This module is responsible for analyzing monitored data. To do so, it takes into account both their current usage by the KNFs composing an SFC and the total resources of each type assigned to them.

The *Infrastructure Status Analyzer* module allows HADES to be aware of the most demanded nodes or of those that should be excluded from some deployments. This concept is based on the premise that some VNFs/KNFs will need features presented in some nodes (e.g., a video camera, or a network card), while for other VNFs/KNFs, such a node is only used for computation. In this way, the resources of these nodes remain available for the network functions that can only be executed inside them. This process can be carried out automatically by HADES if it detects that a particular VNF needs to be executed on a particular node(s) or included as a rule by the infrastructure administrator. The processing of the infrastructure's status information also allows to apply auto-scaling techniques [38].

This module also facilitates the configuration of an application service (e.g., a web service) by adapting the application service to the capabilities of the infrastructure, and vice versa. For this purpose, it uses the algorithms *Application Variability Adapter* (AVA) and *New Devices Finder* (NDF). These independent but complementary algorithms are presented in [9] and have been incorporated as modules of HADES. Sometimes, the infrastructure capabilities cannot be modified (i.e., nodes cannot be added or modified) and it is the software that has to adapt to the infrastructure's capabilities. In this scenario, HADES uses the AVA module to prevent services not supported by the infrastructure from being configured. It achieves this by tailoring the application variability model to match the characteristics of the infrastructure, including both hardware and software aspects. For this purpose, the AVA module receives as input the application's FM and the hardware and software information of the infrastructure and follows the next steps: (1) checks if the available infrastructure supports the infrastructure's requirements of each feature; (2) determines the maximal and minimal values of each attribute supported; and (3) iterates each application's service and device, evaluating and returning the maximal capabilities supported by each device in case of executing the service on it. As a result, the AVA returns the supported and

non-supported features and the value range of the attributes, thus determining the functionalities that the infrastructure supports and the maximum QoS.

At other times, it is necessary to deploy a set of VNFs with specific characteristics (i.e., it is not possible to configure or adapt them to the current capabilities of the infrastructure), but it is possible to extend the capabilities of the infrastructure. In this scenario, HADES uses the NDF module to obtain a description of the new nodes to be included in the infrastructure to support such deployment. The NDF receives as input the configured FM of the application and the information of the infrastructure and follows a two-step process. In the first step, the NDF checks the application functionalities already supported by the infrastructure and verifies that the infrastructure nodes have resources enough to allocate the supposedly supported services. In the second step, the result of the first step is post-processed, identifying the non-supported functionalities and creating a set of provisional devices from the non-supported functionalities. Finally, the NDF operates with this set of provisional devices. As a result, the NDF returns the minimal set of devices needed to support the configured application given as input. The logical models of the AVA and NDF modules, widely detailed in [9], are solved using a satisfiability modulo theories solver. The logic of both modules, as well as the information needed for their execution, are stored in the central knowledge module. In this sense, the infrastructure status analyzer component (label A), which receives monitored data from the resource monitor, is able to foresee if it necessary to scale up the infrastructure (both vertical and horizontal scaling). The NDF algorithm informs about the (new) devices needed to run a specific VNF/SFC. Additionally, the evolution of a VNF/SFC and/or infrastructure may affect the feasibility of the currently deployed services. Some VNFs may require devices with specific characteristics (in terms of location, configuration, computation capabilities, etc.) that the current infrastructure does not meet. The NDF algorithm detects all VNF and SFC's requisites non-supported currently by the infrastructure and returns a minimal configuration (i.e., the minimal set of features and minimal value of the attributes) of the minimal set of nodes needed to deploy the VNF properly, which can be solved by scaling up (vertically or horizontally) the infrastructure.

The HADES dashboard shows all the information tracked by this module and is implemented as a web service using Vuejs.

4.3. Deployment Configurator and Scheduler

This module has two main components: the *Deployment Configurator* and the *energy effIcient TAsk and REsource Allocator* module (iTAREA).

In the *Deployment Configurator*, the infrastructure manager first selects the VNF descriptors of the VNF/KNF from the catalogue (through the OSM API interface). Once selected, the VNF/KNF instances to be deployed are configured using the application FM using the dashboard. Commonly used in software product lines, FMs formally represent, in terms of features, which elements of a family of software products are common, which are variable, and the relationships between them. FMs are represented as a set of hierarchically ordered features composed of parent-child relationships and constraints representing the relationships among features. VNFs/KNFs are configured by setting parameters through the selection of their internal functionalities in the FM. It has been proven that the use of FM facilitates the adoption of edge computing [17, 9]. In addition, many of the tasks resulting from decomposing an IoT application are often common and recurrent, thus fitting very well within an SPL approach. The number of VNFs/KNFs (or tasks) that compose an SFC supporting an application and their characteristics will depend on the configuration of the application FM. If the infrastructure manager wishes to deploy an application that has already been previously configured (e.g., a streaming service), an already configured FM should be selected and this step would be skipped.

Once the application to deploy is configured, the iTAREA is the component responsible for deciding where should be executed each VNF/KNF and the amount of RAM (Mb), disk (Mb), bandwidth, and percentage of CPU that the nodes must allocate to guarantee a certain QoS. This module aims to minimize energy consumption while assuring QoS and is based on the premise that some nodes are more energy efficient than others, that is, some nodes are preferable because they are powered by renewable energy. The energy model used to estimate the energy footprint of each task is contained in the *VNF/KNF Placement Knowledge* module explained later, so the iTAREA module is decoupled from the energy model. To clarify the terminology, iTAREA names VNF/KNF as tasks, as their placement is addressed as a task assignment problem.

$$
\begin{array}{ll}
\textbf{Input:} & N, Tr, \tau, nodesToAvoid, CPUPartitions, ct \\
\textbf{Minimize:} & energy \rightarrow \forall (n,m)\epsilon N, i\epsilon\tau : x_{i,n} eCompu(i,n) + \\
& \quad \sum_{j\epsilon\tau} (h_{i,j} eSend(c_{i,j}, n, m, ct) + x_{j,m} eRecp(c_{i,j}, n, m, ct)) \\
\textbf{Subject to:} & N = N \setminus nodesToAvoid \quad [1] \\
& \forall i\epsilon\tau : \sum_{n\epsilon N} x_{i,n} = 1 \quad [2] \\
& \forall n\epsilon N : \sum_{i\epsilon\tau} x_{i,n} i_{RAM} \leq n_{RAM} \quad [3] \\
& \forall n\epsilon N : \sum_{i\epsilon\tau} x_{i,n} i_{disk} \leq n_{disk} \quad [4] \\
& \forall n\epsilon N : \sum_{i\epsilon\tau} x_{i,n} i_{bandwidth} \leq n_{bandwidth} \quad [5] \\
& \forall n\epsilon N, i\epsilon\tau : x_{i,n} RAM_{i,n} \geq i_{RAM} x_{i,n} \quad [6] \\
& \forall n\epsilon N, i\epsilon\tau : x_{i,n} disk_{i,n} \geq i_{disk} x_{i,n} \quad [7] \\
& \forall n\epsilon N, i\epsilon\tau : x_{i,n} bandwidth_n \geq i_{bandwidth} \quad [8] \\
& \forall n\epsilon N : \sum_{i\epsilon\tau} x_{i,n} CPU_{i,n} \leq CPUPartitionsCores_n \quad [9] \\
& \forall n\epsilon N, i\epsilon\tau : CPU_{i,n} \epsilon \mathbb{N} \quad [10] \\
& \forall n\epsilon N, i\epsilon\tau : x_{i,n} \wedge meets(n,i) \vee \neg x_{i,n} \quad [11] \\
& \forall tr\epsilon Tr, (n,m)\epsilon N, i\epsilon\tau : x_{i,n} tCompu(i,n) + \\
& \quad \sum_{(j)\epsilon tr} h_{i,j} x_{j,m} tCommu(c_{i,j}, n, m, ct) \leq time_{tr} \quad [12] \\
\textbf{Return:} & \forall i\epsilon\tau, n\epsilon N : x_{i,n} \rightarrow RAM_{i,n}; x_{i,n} \rightarrow disk_{i,n}; \\
& x_{i,n} \rightarrow bandwidth_{i,n}; x_{i,n} \rightarrow CPU_{i,n}
\end{array}
\tag{1}
$$

Implementation: As the CPU assigned to each task depends on the computing capacity of the node, the variable *CPUPartitions* contains the portions of CPU that can be assigned to each task by each core; this reduces the search range (and therefore the problem complexity). Therefore, if this variable has a value of 10, the CPU will be allocated in portions of 0.1 cores (100 millicores). Concerning QoS assurance, application services are supported by an SFC composed of a set of tasks or VNFs with time constraints (i.e., they must be completed within a maximum time), which are part of the problem constraints.

This module receives as input the information of the nodes (set N), the tasks (τ), the nodes to exclude (*nodesToAvoid*), the set of time restrictions (Tr), the CPU portions (*CPUPartitions*), and the connection type (ct). As output, it returns the task assignment and the amount of resources that the node must reserve for each assigned task that minimizes the energy consumption assuring a certain QoS (using the energy and latency models stored at the *VNF/KNF Placement Knowledge* module presented in Section 4.5).

The iTAREA module, formalized in Equation 1, does the following (Table 2 constains the definition of variables and functions): first, it excludes the

Table 2: Definition of variables and functions

Variable/Function	Definition
α_n	Percentage of idle energy consumption of node n
τ	Tasks
$v_{i,n}$	Percentage of CPU allocated to task i in node n
$eIdle_n$	Energy required by node n to remain active
$eRecp(c_{i,j}, n, m, ct)$	Energy required by node m to receive $c_{i,j}$ from node n using ct
$eSend(c_{i,j}, n, m, ct)$	Energy required by node n to transmit $c_{i,j}$ to node m using ct
$c_{i,j}$	Data to transmit between tasks i and j
CPU_n	Cycles per second of node n
$CPUPartitions$	Portions of CPU considered
ct	Connection type
$eMax_n$	Maximum energy consumption of node n
ew_n	Energy weight of node n
$h_{i,j}$	1 if tasks i and j are assigned to different nodes, 0 otherwise
$i_{bandwidth}/i_{disk}/i_{RAM}$	Bandwidth/disk/RAM required by task i
$meets(n, i)$	1 if node n meets the requirements of task i
N	Nodes
$n_{bandwidth}/n_{disk}/n_{RAM}$	Bandwidth/disk/RAM available in node n
$nodesToAvoid$	Nodes to avoid from the solution
$Cores_n$	Number of cores of node n
pR	Probability of retransmission
$RAM_{i,n}/Disk_{i,n}$	RAM/disk assigned to tasks i in node n
R_n^{Tx}/R_n^{Rx}	Transmission/reception speed of node n
$tCommu(c_{i,j}, n, m, ct)$	Time required by node n to transmit $c_{i,j}$ from node n to m using ct
$tCompu(i, n)$	Time required by node n to compute task i
$T_{n,z}^{prop}$	Propagation delay between nodes n and z
$time_{tr}$	Maximum time to complete the time restriction tr
Tr	Time restrictions
w_i	Number of CPU cycles of task i
$x_{i,n}$	1 if task i is assigned to node n, 0 otherwise

nodes to be avoided from the set N (1); (2) checks that each task has a node assigned; (3-5) ensure that nodes have RAM, disk, and bandwidth enough to execute their assigned tasks, while (6-8) assure to assign enough resources to properly execute each task; (9-10) check that the nodes do not allocate more CPU than they have available (limiting the search range according to the *CPUPartitions* variable); (11) checks that the nodes fulfil the task requirements (in terms of peripherals and sensing units); (12) verifies the fulfilment of the time restrictions, considering execution and communication times; finally, the iTAREA returns the solution i.e., task assignment, RAM, disk, bandwidth and CPU assigned to each task.

Most approaches provide a solution for the placement problem (that are proven to be NP-hard [39]) using heuristic algorithms or through ILP/NLIP, MILP/MINLP, and SMT (Satisfiability Modulo Theories) solvers [4, 16]. While heuristic algorithms do not ensure to return a solution (even if it exists), ILP/NILP, MILP/MINLP and SMT solvers always return the solution or conversely report the infeasibility of the problem. ILP/NILP and MILP/MINLP solvers differ in the type of variables and the degree of equa-

tions supported: in ILP/NLIP, all variables are constrained to assume integer values, and equations are required to be linear for ILP or potentially non-linear for NLIP; MILP and MINLP can support other types of variables, but at least one variable must be integer-valued, supporting equations being linear/non-linear respectively. Finally, SMT solvers do not restrict the type of variables and support nonlinear equations.

Considering the constraints imposed by each type of solution and the problem at hand, heuristics algorithms, MILP/MILNP and SMT solvers could be adopted (ILP and NILP are discarded for restricting variables to integers). In our work, we chose to use an MILP solver and specifically the *Gurobi Paralleled Mixed Integer Programming* solver. The facts for this choice are: (1) Equations included in the iTAREA algorithm are linear, so using nonlinear solvers would add meaningless complexity; (2) unlike heuristic algorithms, MILP solvers always return a solution, which guarantees that the deployment is feasible or the impossibility to deploy the application if no solution is found; (3) MILP solvers provide solutions significantly quicker than SMT solvers [40]; and (4) Gurobi is especially strong in solving MILP problems [41], supporting multi-threading and aiding problem scalability. Nevertheless, it could be possible to use other solutions to our approach.

4.4. Deployment Manager

This module supports the management (instantiation, modification, and termination) of VNF/KNF deployments.

The number of VNFs/KNFs (or tasks) that support the application and their characteristics will depend on the configuration of the application FM.

For instantiating the VNF/KNF after the iTAREA solution, this module receives the desired deployment from the previous module, processes this information and sends OSM-understandable deployment files to the OSM to carry out the deployment. This means that for replacing OSM with another network orchestration platform is only necessary to modify this module that translates the iTAREA deployments in a format understandable for the new platform.

The VNF/KNF *Deployment manager* also allows both the modification and termination of existing deployments. The elimination of deployments causes the network services inside an application to momentarily stop running and the nodes to release the resources allocated to its execution. The modification of deployments allows extending, reducing, or modifying the

functionality of applications, keeping intact those parts of the application that are not involved.

Deletion is a relatively simple process: the infrastructure manager selects a deployment from among the active ones and orders to delete it, while, the amount of available resources for each node is updated. In the case of modification of an existing deployment, the VNFs/KNFs whose functionality has been modified are deleted and the new VNFs/KNFs are deployed instead. Sometimes, it is not necessary to remove existing tasks, and new tasks are added. Application modifications are carried out by configuring the application FMs. Once completed, the resulting SFC (formed by a combination of new and old VNFs/KNFs) will remain connected following the location log maintained at the backend (see Section 4.4).

4.5. VNF/KNF Placement Knowledge

This component maintains the information retrieved from the KNF/VNF catalog stored by OSM (see figure 1), the KNF/VNF configuration model, up-to-date information about the infrastructure's capabilities (in terms of hardware and software specifications), the energy and latency models (widely described below), and keeps a time series data of the infrastructure's status. This component persistently stores the information needed by the rest of the components, including the logic of our modules AVA, and NDF presented in [9], used by the *Infrastructure Status Analyzer*.

The *VNF/KNF Placement Knowledge* component also contains the location information of each task deployed by HADES and their allocated resources (CPU, memory, disk, and bandwidth). This allows keeping a record of the location of the VNFs/KNFs, assuring the correct communication between them. If the location of a task changes (e.g., because there has been a change in the location of a task or it has been deleted or modified), this information is updated. This mode of operation is based on the Mediator design pattern [42].

All this information is persistently stored in a relational database.

Energy consumption and latency models: The knowledge component also contains the energy consumption and latency models, used to estimate the EC and latency of the application execution, to minimize EC, and to ensure the QoS.

Equation 2 shows the expressions used to calculate the computation time and data transmission [16, 43]:

$$tCompu_{i,n} = \frac{w_i}{CPU_n \upsilon_{i,n}}$$
$$tCommu_{i,j,n,z,ct} = \frac{c_{i,j} + c_{i,j} pR}{Min(R^{Tx}_{n,ct}, R^{Rx}_{z,ct})} + t^{prop}_{n,z} \quad (2)$$

where the computation time of task i in node n ($tCompu_{i,n}$) is obtained by the ratio between the number of CPU cycles (w_i, collected using tools like Linux *perf*) required by the task i and the CPU frequency of the node per core (CPU_n) multiplied by the percentage of CPU allocated to the task on the node ($\upsilon_{i,n}$). Meanwhile, the communication time is calculated by the ratio between the amount of data to transmit between tasks i and j ($c_{i,j}$)–being pR the probability of retransmission due to packet loss–and the transmission/reception speed between nodes (R^{Tx}, R^{Rx})–using connection type ct (*e.g., WiFi 2.4, 4G*). The propagation delays $t^{prop}_{n,z}$ is set as the half of the mean round trip time obtained by pinging from n to z and considered like constant [16].

Equation 3 shows the expressions used to estimate the EC. EC at the nodes is influenced by several factors, such as CPU, storage, and RAM usage, being the CPU the most influential factor and the most commonly used in the literature [16]. The first expression of Equation 3 calculates the computational EC (J) required by the node n to compute task i ($eCompu_{i,n}$), while the second expression represents the idle EC of a node ($eIdle_n$). The energy consumed by task i to communicate with task j is shown in the third and fourth expressions of Equation 3, which calculate the EC for sending and receiving data at nodes n and z (transmitting and receiving nodes, respectively) [16]:

$$eCompu_{i,n} = (1 - \alpha_n)(eMax_n \upsilon_{i,n} + eIdle_n) tCompu_{i,n} ew_n$$
$$eIdle_n = \alpha_n eMax_n$$
$$eSend_{c_{i,j},n,z,ct} = x_{i,n} h_{i,j} P^{Tx}_{n,ct} \frac{c_{i,j} + c_{i,j} pR}{Min(R^{Tx}_{n,ct}, R^{Rx}_{z,ct})} ew_n$$
$$eRecp_{c_{i,j},n,z,ct} = x_{j,z} h_{i,j} P^{Rx}_{z,ct} \frac{c_{i,j} + c_{i,j} pR}{Min(R^{Rx}_{n,ct}, R^{Tx}_{z,ct})} ew_z \quad (3)$$

where $eMax_n$ is the EC of node n when it is fully-utilized (in terms of CPU); υ the CPU utilization ratio (from 0 to number of node cores); and α, whose value is between 0 and 1, represents the percentage of the idle EC of the node (e.g., 30%) [16]–0 if the node was on before the operation. This model is based on the observation that the EC is linear to the CPU utilization ratio which

depends on the computation load [16, 28]. The third and fourth expressions of Equation 3 denote the EC for data sending and receiving respectively from node n to z [16]. P^{Tx} and P^{Rx} represent the transmission powers, while R^{Tx} and R^{Rx} denote the transmission rates for data transmitting and receiving. $x_{i,n}$ is 1 if task i is assigned to node n, 0 otherwise; $h_{i,j}$ is 0 if tasks i and j are assigned to the same node, 1 otherwise. The variable ew (energy weight), whose value is between 1 and 0, represents the importance of energy saving in each node. The higher its value, the more important it is to save energy at that node. This makes it possible to adapt the deployment solution to the needs of the infrastructure. Therefore, unlike other approaches that have a set of nodes on which the reduction of EC is focused [16], our approach allows to customize the importance of energy savings according to the needs of the infrastructure. Thus, if the intention of the infrastructure manager is to focus the reduction of EC on some specific devices or, on the contrary, to execute the greatest number of requests on others, it is sufficient to set this variable to 1 on the devices on which the energy saving is focused and to 0 on the others. This weight-based system allows to provide flexibility to the deployment solutions and eases the definition of policies according to the infrastructure needs. An example of practical use would be to prioritize the execution of tasks on devices powered by solar panels, therefore reducing the whole carbon footprint of the infrastructure.

5. Evaluation

This section first presents a case study, an AR application used to perform the evaluation of HADES on a physical testbed infrastructure supporting heterogeneous scenarios. Then, the reduction in energy consumption is evaluated by deploying the AR application with HADES with respect to the default OSM deployment in four different real scenarios. In addition to this, the energy reduction achieved with HADES is also evaluated in simulated environments, comparing the energy consumption obtained with our approach against the default OSM deployment and four other allocation policies. HADES energy overhead is also evaluated, and the scalability is analyzed using a benchmark.

Case study: Augmented Reality Services. AR is one of the disruptive technologies that most benefit from the 5G and Edge Computing infrastructures, so we present a case study from the AR domain [44]. As a case study, we have implemented *PlanetARium*, an application that uses AR services

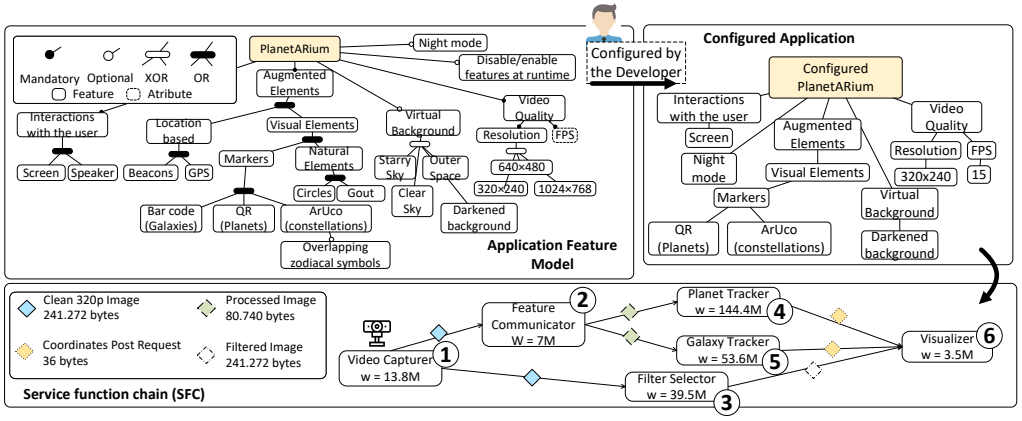

Figure 2: FM of PlanetARium (upper left corner); configured feature model (upper right corner); resulting SFC (bottom)

to display virtual planets, constellations, etc., according to the information obtained from a camera. This application, which was developed for academic purposes, is fully functional: it combines real and virtual objects in a real environment; it runs interactively and in real-time; and it registers (aligns) real and virtual objects with each other, including all the key components of AR systems [45]. AR applications are widely used in edge computing approaches due to their inherent collaborative properties in terms of uplink data collection, computing at the edge, and downlink data delivery. In addition, AR applications are computationally intensive and delay-sensitive, and processing all the information on devices with low computational capacity is prohibitively expensive because it would affect users' expectations in terms of QoS [45, 16]. All these features are gathered in PlanetARium (explained in detail below), which has been chosen as a representative example of the AR family of applications developed using a software product line. However, HADES allows the deployment of any type of application.

The FM of PlaneARium is shown in the upper left corner of Figure 2. One of the main variability points of the application is the feature *Augmented Elements*, which models the elements that will trigger the virtual elements. The application distinguishes between two main methods of displaying virtual elements: based on the location of the device and according to the elements obtained through the camera. The device's location can be determined by the GPS of the device or through beacons (in the first case the device would need a GPS, while in the second case, the device would require Bluetooth).

On the other hand, the information perceived by the camera can be natural elements (everyday objects, of which circles and squares can trigger virtual elements) and markers, such as QR codes (which will display planets) and ArUcos (showing constellations). Thus, the PlanetARium FM represents all the essential elements of an AR system [45]: the inputs, that involve the devices' sensors used to capture the information; the processing features, that are the collection of functions responsible for enriching the reality; and the outputs, which involves projecting virtual content onto the current perspective of the actual environment.

The configured FM of the application used in this work (i.e., a tree with only the features selected) is shown in the upper right corner of Figure 2. It shows the virtual elements will be displayed through a screen, and the background will be that of a dark night. The elements that trigger the virtual elements are markers, that will show planets (QR codes) and constellations (ArUcos). Finally, the resolution chosen is 320 x 240 pixels and the refresh rate is 15 frames per second (FPS). PlanetARium functionality is divided into six different but interrelated tasks of an SFC, which can be placed in different locations. The resulting SFC, composed of six interconnected micro-services, is modelled as a directed graph [4] and shown at the bottom of Figure 2. It encompasses the two discernible types of task dependencies presented in applications: sequential and parallel. In sequential dependencies, one task must finish for the next one to start (KNFs 1-2-4-6; KNFs 1-2-5-6; KNFs 1-3-6), while parallelizable tasks are independent of each other and can be executed concurrently (KNFs 4, 5, and 3; KNFs 2 and 3). Managing these dependencies is further complex when ensuring QoS. Such QoS requirements (video quality features of the FM shown in the upper left corner of Figure 2) are part of the application configuration, allowing us to evaluate how our solution deals with QoS when both sequential and parallel dependencies are presented in the application. Concretely, the KNFs that conform the chain are the following[1]:

- *Video Capturer (KNF 1)*: captures video from a camera and adapts its resolution to the one selected by the user. This KNF requires a node with a camera.

- *Feature Communicator (KNF 2)*: receives the video stream and converts

[1]PlanetARium repository: `https://hub.docker.com/u/planetariumapp`

to black-and-white video for easy element detection.

- *Filter Selector (KNF 3)*: applies a filter to reduce the brightness of the received video and a blur, assimilating the image to a night sky for a more immersive experience.

- *Planet Tracker (ArUco Detector, KNF 4)*: searches for ArUco codes in the received video, identifying each one and determining its coordinates.

- *Galaxy Tracker (QR Detector, KNF 5)*: searches for QR codes in the received video. Once identified, it obtains their coordinates and sends them to the visualizer.

- *Visualizer (KNF 6)*: receives the processed video stream and the coordinates of planets and galaxies. It then overlays the corresponding virtual elements on the video and creates a server accessible through a browser.

All tasks run as Docker containers and communicate through sockets. For video capture and element detection, PlanetARium uses the open source tool OpenCV [46] for Python 3, while for 3D element rendering NodeJS and Three JS. Concerning their requirements in terms of RAM and disk, they are obtained by using the Linux command *htop* and Docker Stats [47]. To ensure the desired QoS, we extract the computational cost and amount of data to transmit for one iteration of each task. Knowing the time it takes for the deployment to perform a complete iteration (i.e., to execute each task once) allows us to know the number of iterations per second (15 in our case, 0.067 seconds per iteration as maximum). Our model uses the number of CPU cycles to determine the computational load of the VNFs/KNFs (see Equations 2 and 3). However, the same task may require a different number of cycles when executed in different devices and, to a lesser extent, vary between iterations (e.g., due to different inputs). Thus, we measure 2,000 iterations of each task (using different inputs) for each node and establish an average number of CPU cycles and an average amount of data transmitted. Specifically, the CPU cycles of KNFs 4-6 and the data to be transmitted from KNFs 4-5 to KNF 6 may depend on the presence of ArUco and QR codes (marker-based tracking [45]). Therefore, the computational load and data to be transmitted of such tasks have been evaluated considering images (captured by KNF 1) with and without the presence of ArUco and QR codes,

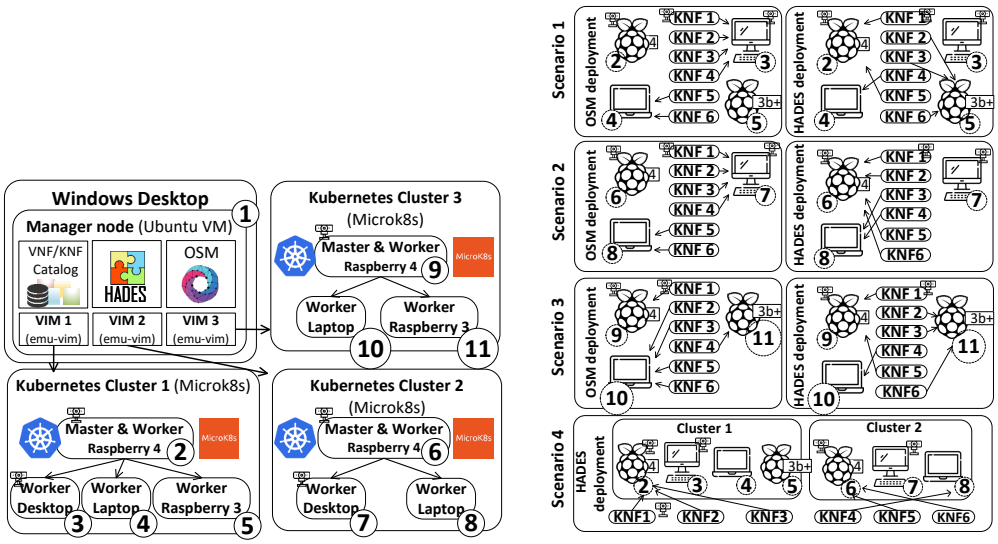

Figure 3: Experimentation setup (left) and PlanetARium deployment (right)

considering different combinations of them in the same image. Furthermore, pictures have been taken in varying lighting circumstances, attempting to represent along with the aforementioned, the different conditions that may affect the computational and communication requirements of PlanetARium KNFs 1-6. In this way, the bias that could exist by considering only one type of input for each task is eliminated. We use the Linux tool *perf* to obtain the computational cost and a traffic sniffer to get the amount of data transmitted between nodes. PlanetARium KNFs do not reserve or limit node bandwidth.

5.1. Energy saving evaluation

Infrastructure. The left side of Figure 3 shows the edge infrastructure used in this work. The infrastructure consists of eleven heterogeneous nodes: one of them operates as a manager node while the remaining nodes are distributed in three Kubernetes clusters (using Microk8s release 1.23). Microk8s release 1.23 is a lightened version of Kubernetes that supports a wide range of architectures and operating systems while reducing the amount of resources required compared to the standard version of Kubernetes, decreasing EC and improving the applicability of the proposal. Nevertheless, HADES is compatible with other lightened versions of Kubernetes as k3s in its current version and could be adapted to support other orchestration solutions if

Table 3: Technical specifications of the infrastructure

Cluster	Node	Role	Type	CPU	Memory	Operating system
-	1	Manager	Desktop	Core i7-7700K (4.2 GHz) (2 cores dedicated to the VM)	8 Gb DDR4 (2400 MHz)	Ubuntu 20.04 LTS (virtualized)
1	2	Master & worker	Raspberry Pi 4	Broadcom BCM2711 Quad core Cortex-A72 (1.5 GHz)	4 GB LPDDR4 (2400MHz)	Ubuntu Mate 20.04.1
	3	Worker	Desktop	Intel Core i5-4440 (3.10 GHz)	12 Gb DDR3 (1333 MHz)	Ubuntu 20.04 LTS
	4	Worker	Laptop	Intel Core i5-5300U (2.30 GHz)	4 GB DDR3 (1600 MHz)	Ubuntu 20.04 LTS
	5	Worker	Raspberry Pi 3 B+	Broadcom BCM2837B0 Quad Cortex-A53 (1.4 GHz)	1 GB LPDDR2 (400 MHz)	Ubuntu Server 20.04
2	6	Master & worker	Raspberry Pi 4	Broadcom BCM2711 Quad core Cortex-A72 (1.5 GHz)	4 GB LPDDR4 (2400MHz)	Ubuntu Mate 20.04.1
	7	Worker	Desktop	Intel Core i5-3570K (3.70 GHz)	12 GB of DDR3 (1333 MHz)	Ubuntu 20.04 LTS
	8	Worker	Laptop	Intel Core i5-5200U (2.20 GHz)	4 GB of DDR3 (1600 MHz)	Ubuntu 20.04 LTS
3	9	Master & worker	Raspberry Pi 4	Broadcom BCM2711 Quad core Cortex-A72 (1.5 GHz)	4 GB LPDDR4 (2400MHz)	Ubuntu Mate 20.04.1
	10	Worker	Laptop	Intel Core i5-5300U (2.30 GHz)	4 GB of DDR3 (1600 MHz)	Ubuntu 20.04 LTS
	11	Worker	Raspberry Pi 3 B+	Broadcom BCM2837B0 Quad Cortex-A53 (1.4 GHz)	1 GB LPDDR2 (400 MHz)	Ubuntu Server 20.04

necessary (see Section 4.4). Table 3 summarizes the technical specifications of the eleven nodes that conform to the virtual and physical infrastructure used for the experiments. The manager node (labelled 1 on the left side of Figure 3 and specified in the first row of Table 3) runs instances of HADES, OSM (its most up-to-date release, THIRTEEN), the VNF/KNF catalogue repository, and three virtualized VIMs (that are virtualized using OSM's native *emu-vim tool*) in an Ubuntu VM hosted on a Windows 10 operating system using VirtualBox.

To carry out an efficient task allocation, HADES uses the idle and busy EC of the nodes. Therefore, the EC of the nodes has been analyzed both at idle (with the Microk8s installed) and during a 1-minute CPU matrix stress test [48]. The matrix stressor is a good way to exercise CPU floating-point operations, as well as the memory and data cache of the processor. To measure the EC we use Watts Up? Pro [49], a physical meter that allows real-time monitoring of the amount of energy supplied to the device connected to the power supply.

Regarding communication, the round-trip time between nodes is obtained by pinging from node a to b (considering the average of sending 1,000 small packets). The transmission rate has been analyzed using *iPerf* [50] tool (considering the average transmission rate of sending packages during 100 seconds). Finally, the power transmission is given directly from the characteristics of the nodes' network cards.

Table 4: Deployment of PlanetARium with OSM by default (left) and HADES (right)

		OSM						HADES					
		KNF 1	**KNF 2**	**KNF 3**	**KNF 4**	**KNF 5**	**KNF 6**	**KNF 1**	**KNF 2**	**KNF 3**	**KNF 4**	**KNF 5**	**KNF 6**
Scenario 1	**Node**	3	3	3	3	4	4	2	5	5	4	2	5
	CPU limits (m)	-	-	-	-	-	-	400-500	400-500	600-700	1400-1500	1200-1300	400-500
	RAM limits (Mb)	-	-	-	-	-	-	32-42	53-63	42-52	53-63	42-52	42-52
	EC[a]/REC[b]	**14.3 W / -**						**6.5 W / 45.4%**					
Scenario 2	**Node**	7	7	7	7	5	6	6	6	8	8	6	6
	CPU limits (m)	-	-	-	-	-	-	400-500	300-400	200-300	1400-1500	1200-1300	200-300
	RAM limits (Mb)	-	-	-	-	-	-	32-42	53-63	42-52	53-63	42-52	42-52
	EC/REC	**14.7 W / -**						**7.9 W / 46.7%**					
Scenario 3	**Node**	9	10	10	11	10	10	9	11	11	10	9	11
	CPU limits (m)	-	-	-	-	-	-	400-500	400-500	600-700	1400-1500	1200-1300	400-500
	RAM limits (Mb)	-	-	-	-	-	-	32-42	53-63	42-52	53-63	42-52	42-52
	EC/REC	**The deployment does not meet the QoS**						**The OSM deployment does not meet the QoS**					
Scenario 4	**Node**	-	-	-	-	-	-	2	2	2	8	6	6
	CPU limits (m)	-	-	-	-	-	-	400-500	300-400	400-500	1400-1500	1200-1300	200-300
	RAM limits (Mb)	-	-	-	-	-	-	32-42	53-63	42-52	53-63	42-52	42-52
	EC/REC	**Not supported**						**7.2 W / [49.6%,51%]**					

[a] EC: Energy consumption
[b] REC: Reduction in energy consumption (respecting the default OSM deployment)

5.1.1. Experimentation in physical scenarios

To analyze the reduction in EC obtained through our proposal, this section compares the EC of the default Kubernetes scheduler selection with the selection provided by HADES for the infrastructure and application presented in Sections 5 and 5.1. For this purpose, four different scenarios for deploying PlanetARium are considered: Scenario 1 considers only the nodes in Cluster 1; the second scenario considers only the nodes in Cluster 2; Scenario 3 considers only the nodes in Cluster 3; and finally, the fourth scenario considers the complete infrastructure (Clusters 1-3) for deploying PlanetARium KNFs. In Scenarios 1-3 we will apply both the default Kubernetes scheduler (kube-scheduler) and HADES, while in the fourth scenario only can be considered the HADES solution, as OSM forces the deployment KNFs of the same SFC in the same cluster. Finally, we will measure the EC of the nodes before and after deploying the application, thus obtaining the deployment EC. To measure the QoS of the deployment to check if it complies with the QoS requirement of 15 FPS, an on-screen frame counter is introduced. For a fair comparison of results, all tests were performed at the same location and on the same day, under the same temperature conditions and node workload status.

The right side of Figure 3 shows the resulting KNF assignments and Table 4 resumes the deployment decision and resource allocation taken by the Kubernetes scheduler in OSM by default (columns grouped under OSM on the left side) and HADES (right). Note that the Kubernetes scheduler (used by the OSM's orchestrator) does not limit RAM or disk use by default [11].

Scenario 1 (Cluster 1): kube-scheduler assigns KNFs 1-4 to Node 3, and KNFs 5-6 to Node 4. HADES opts for assigning KNFs 1 and 5 to Node 2, KNFs 2-3 and 6 to Node 5, and KNF 4 to Node 4. Note that HADES assigns KNF 1, which requires a camera, to Node 2 because the Node 2 camera supports video capture at 15 FPS (the minimal QoS desired); the same check would be performed for any other peripheral that could compromise the QoS. Regarding the EC, the base consumption (before the application deployment) of Cluster 1 is 77.9 W, being 92.2 W after the PlanetARium deployment. Therefore, the application EC with kube-scheduler is 14.3 W (i.e., 15.5% more). When deployed with HADES, the EC after the deployment drops to 85.7 W, which means an application EC of 7.8 W (i.e., 9% more). Regarding QoS, both deployments meet the non-functional requirement of 15 FPS. To achieve this, it is crucial to allocate the correct amount of resources to each KNF. Lines 2-5 of Table 4 (right) contain the resource allocation of HADES for Scenario 1. Note that Kubernetes does not set a maximum resource usage margin (so QoS cannot be assured as node workload increases), as HADES does.

Scenario 2 (Cluster 2): While kube-scheduler deploys KNFs 1-4 in Node 7 and KNFs 5-6 in Node 8, HADES assigns KNFs 1-2, 5-6 to Node 6 and KNFs 3-4 to Node 8. The base EC of this cluster is 76.9 W, increasing to 91.6 W (i.e., 15.5% more) after the deployment using kube-scheduler (see the right side of Figure 3. Thus, the application EC is 14.7 W. As for deployment with HADES, the EC is 84.8 W, resulting in an application EC of 7.9 W (i.e., an energy footprint of 9.3%, minor than kube-scheduler selection). Concerning QoS, both deployments meet the QoS of 15 FPS established in the configuration phase. Lines 6-9 of Table 4 (right) show the HADES resource allocation in more detail.

Scenario 3 (Cluster 3): kube-scheduler assigns KNF 1 to Node 9, KNFs 2-3, 5-6 to Node 10 and KNF 4 to Node 11. HADES opts for assigning KNFs 1 and 5 to Node 9, KNFs 2-3 and 6 to Node 11, and KNF 4 to Node 10 (see the right side of Figure 3). The base EC of Cluster 3 is 14.9 W, which grows to 20.7 W with the default OSM deployment, meaning an application EC of 5.8 W. With HADES, the EC after deployment is 22.6 W, which represents an application EC of 7.7 W, an increase of 1.9 W over the default OSM deployment. Nonetheless, the comparison between the two deployments is not fair, as the Kubernetes deployment does not meet never the expected QoS (FPS fluctuates between 3 and 11). This is because Kubernetes assigned the most CPU-demanding task (KNF 4, see Figure 2) to the least CPU-powerful

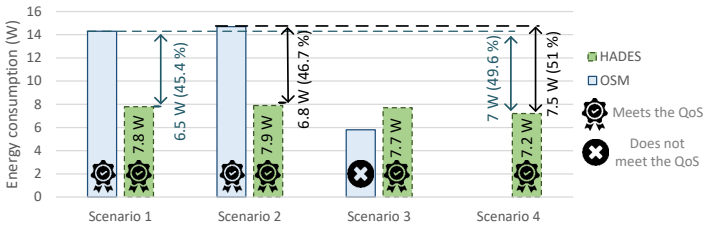

Figure 4: Energy consumption for physical scenarios. Natively, OSM does not support deployments considering multiple VIMs (Scenario 4).

node (Node 11), causing a bottleneck. However, despite the HADES solution consuming a little bit more, it meets the expected QoS (resource allocation is shown in Table 4).

Scenario 4 (Clusters 1-3): When considering all the clusters, HADES opts to assign KNFs 1-3 to Node 2 (in Cluster 1), KNFs 5 and 6 to Node 6 and KNF 4 to Node 8 (in Cluster 2) (see the right side of Figure 3). The whole infrastructure, considering all clusters, has a base EC of 169.7 W, which rises to 176.9 W after the deployment of PlanetARium. Consequently, the application consumes 7.2 W. Concerning QoS, the HADES deployment meets the expected frame rate of 15 FPS. Table 4 (right) contains the resource allocation for HADES deployments. In this scenario, and compared with the results of the previous ones, the energy footprint is the least.

Figure 4 collects the EC obtained for the different scenarios. Experiments reveal that the HADES deployments reduce the EC compared with kube-scheduler in Scenarios 1,2 and 4 maintaining the same QoS. Specifically, for Scenario 1 the EC reduction is 6.5 W, which means a decrease of 45.4%. For Scenario 2, the application EC is 14.7 W deploying PlanetARium with kube-scheduler and 7.9 W with HADES, resulting in an EC reduction of 6.8 W (46.7%). In Scenario 4, the EC of the application is 7.2 W, being the most energy-efficient deployment. This represents a 49.6% reduction in EC compared to deployment using kube-scheduler in Scenario 1, and a 51% for Scenario 2–the scenarios in which the frame rate with kube-scheduler is as desired. The EC in Scenario 3 of the HADES deployment is slightly higher (1.9 W) but the allocation of resources made by HADES meets the minimum QoS of 15 FPS required in all cases, while the deployment of kube-scheduler in Scenario 3 did not achieve that purpose, providing a poor QoS.

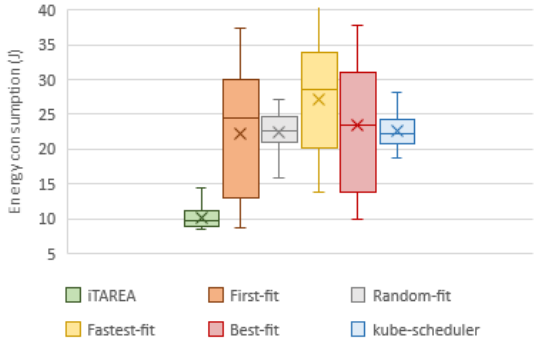

Figure 5: Energy consumption of the deployment solutions using different placement policies.

5.1.2. Experimentation in emulated scenarios

In physical experiments, we are limited to the resources (nodes) we have available. For that reason, this subsection tries to draw further conclusions by applying the iTAREA module to the deployment of applications in emulated infrastructures, evaluating the energy consumption of the same energy model presented in Section 4.3. To this aim, we compare the EC of the deployment using the iTAREA module with (1) the obtained using kube-scheduler (we have obtained the Kubernetes' scoring policy used to assign tasks to nodes from [51]); (2) the policy *Best fit* stated in [52], that selects the node with the most available resources among those capable of running the application; (3) *First fit*, that allocates the application to the first node it finds that meets the application requirements; (4) *Random fit* (also considered in other works [53]), that selects a random node that accomplishes the application's requirements; and (5) *Fastest fit*, which selects for the most powerful node in terms of CPU capable of running the application. The experiments consider applications of between 10 and 30 tasks and infrastructures of between 20 and 30 nodes. The characteristics of the nodes (CPU, RAM, storage, free resources, $eMax$, α, peripherals, etc.) and applications requirements (in terms of CPU, RAM, disk, data to transmit/receive, etc.) have been randomized in each experiment. Each experiment has been performed 30 times[2][3].

Figure 5 illustrates using a box-plot the range of EC (J) resulting from

[2]Source code available at: `https://doi.org/10.5281/zenodo.7712107`
[3]Results spreadsheet available at: `https://doi.org/10.5281/zenodo.7712024`

the application of the different placement policies (iTarea, First-fit, Random-fit, Fastest-fit, Best-fit, default kube-scheduler). Results from experiments reveal that *Fastest-fit* policy (orange box) has obtained the worst energy consumption results. This is because, although not necessarily, the most computationally powerful nodes often tend to be the most energy-consuming [43]. The best energy results are obtained through our energy-aware proposal (green box), with an overall reduction in EC between 54% and 59% when compared to the rest of the assignment policies. Results in EC reduction are quite in line with the energy reduction obtained in practice performed and discussed in the previous section.

Regarding the execution time of the placement, we consider the time it takes to complete one complete iteration of the application (i.e., all tasks that form the SFC): for PlanetARium, it ranges from 0.006 (in case of *Fastest-fit*) to 0.029 seconds (*Random-fit*). The average execution time of the iTAREA module assignment solutions is 0.011 seconds.

5.2. HADES energy overhead

In Section 5.1.1 it has been shown that HADES significantly reduces the energy consumption of running applications. However, keeping HADES instantiated and running has an associated energy consumption. Then, for HADES to be beneficial and feasible such energy overhead must be less than the reduction in energy consumption achieved. This section evaluates this impact on the overall EC.

To conduct this experiment, we evaluate the energy consumption of Node 1 (where HADES instance runs–see the left side of Figure 3) in three scenarios: Scenario (1) idle (without OSM and HADES); Scenario (2) with a clean OSM installation; and Scenario (3) having OSM and HADES installed. The experiments have a duration of one minute. In the case of measurements taken for Scenarios 2 and 3, the experiments include the time required to deploy PlanetARium KNFs. As in the rest of the tests performed in this work, we have used the Watts Up? Pro physical energy meter.

Figure 6 shows the average energy consumption obtained during the experiments. Node 1 has an idle energy consumption (Scenario 1) of 51.6 W. With OSM instantiated and running, this value increases to 53.03 W, which is an increase over Scenario 1 of 1.44 W (2.79%). With OSM and HADES running, the energy consumption of Node 1 is 53.37 W, which is an increase over Scenario 1 of 1.76 W (3.41%), and 0.32 W (0.6%) over Scenario 2.

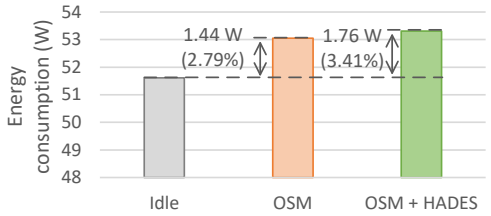

Figure 6: Energy overhead of HADES (Node 1).

Based on the experiments conducted, we can derive that HADES use does not entail a significant energy consumption overhead, especially when compared to the reduction in energy consumption it is able to achieve (see Sections 5.1.1, 5.1.2).

5.3. Scalability

The critical point that threatens the applicability of our proposal lies in the execution time of the iTAREA module. This time depends largely on the nature of the problem and its size [40]. This section evaluates the execution time of the iTAREA module, which runs during the application deployment and migration process, for different problem sizes. With this purpose, we develop a *Benchmark* version of the iTAREA[2], which allows setting the number of VNFs/KNFs of the SFC, the number of nodes that form the infrastructure, and the CPU partitions. A preliminary study revealed that, for infrastructures of less than 20 nodes, the iTAREA returns the solution almost instantly. Thus, the experiments consider infrastructures of 30, 40, and 50 nodes respectively, 5 and 10 CPU partitions per core (100 and 200 millicores allocations), and 10 to 190 VNFs/KNFs. The characteristics of the application (number of CPU cycles, memory, disk, bandwidth, peripherals, etc.) and of the nodes (CPU, cores, memory, disk, bandwidth, peripherals, etc.) are randomly set in each experiment. Since the iTAREA module can be both on a local desktop and in the cloud, we will run the tests both locally (AMD Ryzen 7 5800x, 8 cores, 16 threads at 3.8GHz, 32 Gb RAM) and on the cloud (AMD EPYC 7H12, 128 cores, 256 threads at 2.6 GHz, 64 Gb RAM), using two configurations: (1) using all its potential (128 cores, 256 threads); and (2) using 64 cores (128 threads). Each experiment is performed 30 times[3].

Tables 5, 6, and 7 contain the mean and standard deviation of the iTAREA module execution time when executed in the three infrastructures described

Table 5: Execution times of the iTAREA module (desktop)

KNFs/VNFs			10	30	50	70	90	110	130	150	170	190
30 Nodes	5 Partitions	mean (s)	0.100	0.727	2.161	4.033	10.818	21.907	136.278	342.125	1024.519	651.744
		std	0.016	0.006	0.021	0.043	0.088	0.176	0.384	1.505	17.867	1.792
	10 Partitions	mean (s)	0.117	0.691	1.745	3.240	5.761	10.762	13.956	21.798	46.812	72.535
		std	0.002	0.004	0.008	0.022	0.053	0.038	0.043	0.066	0.058	0.464
40 Nodes	5 Partitions	mean (s)	0.138	1.184	2.736	5.327	14.744	25.864	105.353	964.133	1117.226	517.113
		std	0.005	0.017	0.037	0.062	0.157	0.273	0.696	3.325	1.020	2.202
	10 Partitions	mean (s)	0.152	0.848	2.154	4.317	7.669	12.732	20.417	30.638	37.596	59.984
		std	0,002	0,004	0,008	0,013	0,024	0,048	0,155	0,174	0,116	0,260
50 Nodes	5 Partitions	mean (s)	0.154	1.340	3.254	7.400	13.514	27.561	60.598	227.980	3060.674	25779.594
		std	0,006	0,024	0,071	0,103	0,186	0,158	0,575	2,108	6,276	0,022
	10 Partitions	mean (s)	0.187	1.044	2.845	5.453	9.276	14.073	23.211	29.247	37.163	61.656
		std	0,005	0,015	0,038	0,053	0,069	0,069	0,268	0,238	0,342	0,005

Table 6: Execution times of the iTAREA module (server 64 cores)

KNFs/VNFs			10	30	50	70	90	110	130	150	170	190
30 Nodes	5 Partitions	mean (s)	0.119	1.008	3.080	5.847	11.861	22.685	69.9394	119.500	184.876	211.911
		std	0.001	0.058	0.459	0.489	0.441	1.159	1.215	1.734	5.044	11.033
	10 Partitions	mean (s)	0.143	0.980	2.473	4.455	8.388	14.126	17.991	27.621	39.958	44.883
		std	0.021	0.044	0.087	0.022	0.257	0.448	0.343	0.467	1.224	0.890
40 Nodes	5 Partitions	mean (s)	0.156	1.629	3.543	8.020	15.396	26.140	49.3671	161.515	204.119	367.793
		std	0.002	0.108	0.311	0.172	0.603	1.096	1.534	4.775	1.757	2.518
	10 Partitions	mean (s)	0.204	1.166	2.991	6.312	10.893	18.072	23.606	34.947	43.877	59.635
		std	0,069	0,042	0,024	0,205	0,269	0,370	0,244	0,484	0,536	0,563
50 Nodes	5 Partitions	mean (s)	0.201	1.985	4.661	9.831	19.344	33.057	51.940	112.765	192.171	10587.031
		std	0,067	0,015	0,035	0,020	0,301	0,339	0,411	0,512	0,742	0,003
	10 Partitions	mean (s)	0.234	1.461	3.840	7.407	12.999	19.468	29.264	39.221	48.502	69.824
		std	0,206	0,247	0,282	0,747	1,475	0,766	1,166	6,739	12,453	0,002

Table 7: Execution times of the iTAREA module (server 128 cores)

KNFs/VNFs			10	30	50	70	90	110	130	150	170	190
30 Nodes	5 Partitions	mean (s)	0.118	1.104	2.992	5.665	12.106	24.325	63.766	117.687	178.807	208.284
		std	0.002	0.053	0.143	0.190	0.221	0.938	1.266	0.464	0.661	0.714
	10 Partitions	mean (s)	0.139	1.100	2.623	4.464	8.453	14.742	18.653	28.025	39.735	46.084
		std	0.003	0.064	0.127	0.011	0.356	0.348	0.197	0.412	0.453	0.467
40 Nodes	5 Partitions	mean (s)	0.158	1.791	3.901	8.239	15.510	26.788	47.800	125.015	324.772	190.969
		std	0.002	0.178	0.125	0.316	0.335	1.032	1.297	0.984	1.281	0.554
	10 Partitions	mean (s)	0.186	1.333	3.011	6.562	11.107	18.586	24.770	36.064	45.706	64.567
		std	0,000	0,054	0,009	0,121	0,285	0,419	0,282	0,588	0,227	0,649
50 Nodes	5 Partitions	mean (s)	0.196	2.156	4.798	9.729	19.118	31.649	46.900	98.196	209.074	397.387
		std	0,249	0,169	0,370	0,332	0,381	1,429	0,541	0,636	1,109	19,995
	10 Partitions	mean (s)	0.235	1.568	3.915	7.539	13.299	19.934	30.768	40.385	49.297	75.649
		std	0,069	0,014	0,028	0,227	0,216	0,376	0,328	0,338	0,426	0,424

above (the lowest times for each scenario–nodes and CPU partitions–are highlighted in grey). Experiments reveal that the execution time is shorter when the CPU is allocated in 100-millicore portions (10%, 10 CPU partitions) than when the portions are 200-millicore (5 CPU partitions). For highly heterogeneous infrastructures of 30 to 50 nodes and applications of 30 VNFs/KNFs or less, the iTAREA module returns the solution almost instantly

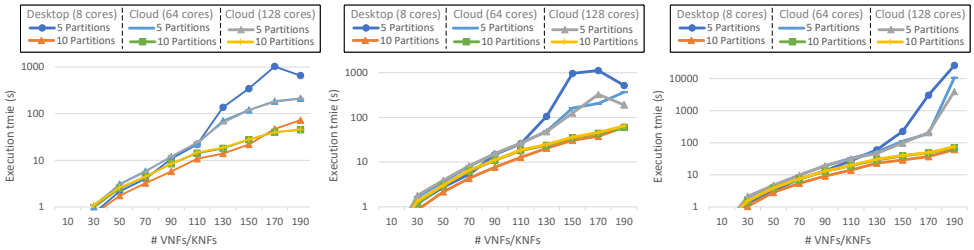

Figure 7: Graphical representation in logarithmic scale of the average execution time of the iTAREA module (Tables 5, 6, and 7) for 30 (left), 40 (middle), and 50 (right) nodes.

(especially in the case of the desktop). Figure 7 employs a logarithmic scale to graphically depict the values from Tables 5, 6, and 7. This representation simplifies the understanding of execution time trends in relation to problem size across different execution infrastructures. In the context of the three considered infrastructures, when handling up to 110 VNFs/KNFs within the iTAREA configuration of 5 CPU partitions (and in almost all scenarios involving 10 CPU partitions), it can be observed that a larger use of resources (in the cloud) results in a greater execution time. This occurs because, on occasion, increasing the number of cores can result in programs running slower in terms of total response time compared to less resource-intensive devices. This is because cloud servers often allocate more time to tasks like communication, control, and load balancing between threads and service requests, which can outweigh the computational benefits gained from their utilization[54].For larger problem sizes, the problem scales well concerning the number of cores, being the cloud up to 6.5 times faster than the desktop in some cases (50 nodes, 5 CPU partitions). Nonetheless, it is important to mention that numerous VNFs have the potential to be utilized across multiple SFCs, only requiring a periodic increase in their instances during deployments to prevent overwhelming (auto-scaling) [55]. This fact reduces the number of VNFs/KNFs to deploy. Therefore, we can conclude that a commercial desktop computer is sufficient to run our proposal and has been demonstrated to be fast enough to be applicable in real-world environments.

6. Threats to validity

This section presents the threats to validity of the evaluation and experimentation presented in the previous section. The four threats are conclusion, internal, construct, and external validity [56].

6.0.1. Conclusion Validity

For the evaluation of the EC reduction of HADES (in Section 5.1.1) we have used a physical measurement tool, Watts Up? Pro. This tool, commonly used by the community, is accepted as a tool that reports accurate EC results [49]. Although all tests have been performed with the same energy meter and under the same conditions, the EC results have not been compared with those obtained with other measurement tools.

6.0.2. Internal Validity

To determine the EC of the physical deployments (Section 5.1.1) and the energy overhead of the proposal (Section 5.2), the difference between the base EC and the energy consumption after deployment was used. Nevertheless, several system processes and applications may be running in the background, so the EC of the application cannot be completely isolated from the rest of the system processes. To mitigate uncontrolled factors, nodes with clean systems were used, and all tests were performed at the same location and on the same day, under the same conditions of temperature and node status.

6.0.3. Construct Validity

The proper functioning of HADES depends to a large extent on the correctness of the nodes (CPU, disk, peripherals, etc.) and applications (CPU cycles, data to transmit, etc.) information. Currently, the process of collecting this information relies in part on the infrastructure manager, and human error can occur. Such an effect could be mitigated by including in HADES the functionality to obtain these values automatically.

To determine the energy consumption overload of our approach (Section 5.2), we use the case study of the application and infrastructure we have available (Section 5.1.1). The deployment of a different application (e.g., formed by more VNFs/KNFs) may result in a further increase in the energy consumption of the proposal. However, the experiments conducted show that the energy consumption of HADES is negligible compared to the benefit obtained, and this difference could increase when deploying larger applications.

6.0.4. External Validity

We deployed an AR application on a heterogeneous infrastructure with three VIMs, considering four deployment scenarios. EC results obtained will depend on the characteristics of the infrastructure and the deployed applications. When testing on physical infrastructures, we are limited to the

resources available. A larger and more heterogeneous infrastructure would help us to obtain more generalized EC results. For this reason, in Section 5.1.2, we extend energy reduction experiments by simulating deployments in fictitious infrastructures. Therefore, we cannot say outright that the results are generalizable to all types of real cases, but, the energy-saving and scalability results are very promising. To mitigate this threat in emulated scenarios we randomize the characteristic of nodes and the application service requirements.

7. Background

This section provides a brief background on MANO reference architecture and overviews existing solutions developed for NFV management.

Since NFV was considered a key technology for the evolution of mobile networks [16], both industry and academia have devoted many efforts to their standardization, adoption and realization. With this aim, the ETSI developed a Reference Architecture Framework, a specification that supports the NFV MANO [57].

In general terms, MANO is responsible for VNF life cycle management (also known as VNF onboarding [58]), from getting a VNF running on an NFV platform (i.e., placement decision, resource allocation, day-0, day-1 and day-2 configuration), and its operation until it is terminated. According to the ETSI-NFV reference architecture, its main components are the NFV orchestrator, the VNF manager, and the VIM. These components undertake the NFV infrastructure management, resource allocation, function virtualization, placement, etc. The management of SFCs is also addressed by the ETSI NFV architecture. The NFV orchestrator is concerned with the management of the life cycle of the SFC and their constituent virtual NFs (instantiation, update, scaling, migration, and termination) in coordination with VNF managers.

Although the ETSI NFV MANO does not define standards for the implementation of these managers, nor the interfaces between them for their coordination, the industry has provided some open source projects that follow the architectural blueprint of the ETSI MANO [59, 60] such as OSM [61], OPNFV [62], ONAP [63], and Cloudfy [64].

As the existing NFV platforms are all very similar in terms of components and services offered, we have compared them in terms of the resources required by the recommended installation. Efficient use of infrastructure resources is essential to our objective, as edge devices are often resource-constrained, and

Table 8: Recommended resources for installing MANO solutions for NFV

Solution	Number of devices		vCPUS	RAM (Gb)	Disk (Gb)
OPNFV [62]	6	1 (Jumphost) 5 others	16 15 (each)	32 32 (each)	250 2 x 500 (each)
ONAP [63]	**1**		112	224	160
Cloudfy [64]	**1**		8	16	64
OSM [65]	**1**		**2**	**8**	**40**

the amount of device resources used is closely related to energy consumption [15]. Table 8 includes details about the capacity of devices they can orchestrate, as well as the necessary specifications for installation such as virtual CPUs, Random-Access Memory (RAM) in gigabytes, and storage in gigabytes. The minimum amounts for each resource type are shown in bold. A comparison of the values given in the official documentation for these platforms led us to choose OSM, as the recommended resources for its installation are lower than for the other solutions. In addition, OSM has the support of many telecommunication operators and providers (such as Bell, Telefónica, T-Mobile, etc., just to mention a few[4]) and an extensive list of organisations providing OSM compatible VIMs, such as Amazon and RedHat[5].

8. Conclusions and future work

NFV and edge computing have established themselves as the key technologies in the adoption of B5G communication technology. Nevertheless, there are still technical challenges to be addressed such as guaranteeing network performance, efficient placement, and energy footprint reduction.

This paper presents HADES, a solution developed to work in conjunction with OSM, extending its functionalities. The result is a platform that optimizes the placement of VNFs and the assignment of resources to minimize the EC, considering the problem of SFC placement, while assuring the QoS. In addition, HADES architecture follows the N-MAPE-K loop defined in the DAEMON project that funds this work, which comprises different modules that allow the placement and management VNFs deployments in edge infrastructures, as well as the monitoring of resources. Our proposal has

[4]OSM members and participants https://portal.etsi.org/TB-SiteMap/OSM/List-of-OSM-Members

[5]https://osm.etsi.org/wikipub/index.php/VIMs

been extensively evaluated to analyze the energy reduction it can achieve, the energy overhead it causes in the infrastructure, and its execution time on a desktop and in the cloud. We have applied HADES to the deployment of AR services in a heterogeneous edge infrastructure and physically measured the energy consumption after deployment, obtaining a reduction in EC of up to 51% compared with the default deployment proposed by OSM. QoS has been measured after deployment, demonstrating that the deployment of HADES met the desired QoS. These experiments have been extended using simulated environments, obtaining up to 59% reduction in EC compared to other five allocation policies. The proposal has been found to have negligible energy overhead compared to the reduction in EC obtained and proved to be fast enough to be applicable in real environments (even when run on commercial computers).

Ongoing and future work includes the incorporation of Scaphandre [66] to monitor the current energy consumption of the KNF after their deployment. Scaphandre is a monitoring agent dedicated to energy consumption metrics that was incorporated very recently into Kubernetes. We are also working on automating the collection of experimental data needed for placement optimization by HADES to enhance the energy model. In addition, we plan to use machine learning techniques to automatically estimate the energy consumption of KNFs running in different machines using different data sets, including the energy consumption reported by Scaphandre, during the network operation. Finally, the outputs of the iTAREA optimal algorithm will be used to train a machine learning model, which will be compared in execution time and accuracy with the iTAREA module.

Acknowledgments

This work is supported by the European Union's H2020 research and innovation programme under grant agreement DAEMON 101017109 and by the projects co-financed by FEDER funds LEIA UMA18-FEDERJA-15 and IRIS PID2021-122812OB-I00 (MCI/AEI).

References

[1] H. Elazhary, Internet of Things (IoT), mobile cloud, cloudlet, mobile iot, iot cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions, Journal of Network and Computer Applications 128 (2018) 105–140. doi:10.1016/j.jnca.2018.10.021.

[2] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, T. Zhou, A survey on edge computing systems and tools, Proceedings of the IEEE 107 (2019) 1537–1562.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (2010) 50–58. URL: https://doi.org/10.1145/1721654.1721672. doi:10.1145/1721654.1721672.

[4] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Communications Surveys Tutorials 19 (2017) 1628–1656. doi:10.1109/COMST.2017.2682318.

[5] V. G. Plc, Sustainable business report, 2019. URL: https://www.vodafone.com/content/dam/vodcom/sustainability/pdfs/sustainablebusiness2019.pdf, online; acc. 10 June 2023.

[6] O. Gheibi, D. Weyns, F. Quin, Applying machine learning in self-adaptive systems: A systematic literature review, ACM Trans. Auton. Adapt. Syst. 15 (2021).

[7] L. Fuentes, M. Amor, Ángel Cañete et al., DAEMON Deliverable 4.2: Refined design of intelligent orchestration and management mechanisms, 2022. doi:10.5281/zenodo.7573188, This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 101017109.

[8] A. Cañete, M. Amor, L. Fuentes, Energy-efficient deployment of iot applications in edge-based infrastructures: A software product line approach, IEEE Internet of Things Journal 8 (2021) 16427–16439. doi:10.1109/JIOT.2020.3030197.

[9] A. Cañete, M. Amor, L. Fuentes, Supporting iot applications deployment on edge-based infrastructures using multi-layer feature models, Journal of Systems and Software 183 (2022) 111086. doi:https://doi.org/10.1016/j.jss.2021.111086.

[10] Reid et al., Osm scope, functionality, operation and integration guidelines, ETSI, White Paper (2019).

[11] Kubernetes' docs, 2023. URL: https://kubernetes.io/docs/home/, online; acc. 10 June 2023.

[12] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, T. Magedanz, Service function chaining in next generation networks: State of the art and research challenges, IEEE Communications Magazine 55 (2017) 216–223. doi:10.1109/MCOM.2016.1600219RP.

[13] K. Kaur, V. Mangat, K. Kumar, A comprehensive survey of service function chain provisioning approaches in sdn and nfv architecture, Computer Science Review 38 (2020) 100298. doi:https://doi.org/10.1016/j.cosrev.2020.100298.

[14] M. Aazam, S. Zeadally, K. A. Harras, Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities, Future Generation Computer Systems 87 (2018) 278 – 289. doi:https://doi.org/10.1016/j.future.2018.04.057.

[15] Y. Mao, C. You, J. Zhang, K. Huang, K. B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Communications Surveys Tutorials 19 (2017) 2322–2358.

[16] Mao et al., A survey on mobile edge computing: The communication perspective, IEEE Communications Surveys Tutorials 19 (2017) 2322–2358.

[17] Cañete et al., Energy-efficient deployment of iot applications in edge-based infrastructures: A software product line approach, IEEE Internet of Things Journal 8 (2021) 16427–16439. doi:10.1109/JIOT.2020.3030197.

[18] C. Jiang, X. Cheng, H. Gao, X. Zhou, J. Wan, Toward computation offloading in edge computing: A survey, IEEE Access 7 (2019) 131543–131558. doi:10.1109/ACCESS.2019.2938660.

[19] D. Li, P. Hong, K. Xue, J. Pei, Virtual network function placement and resource optimization in nfv and edge computing enabled networks, Computer Networks 152 (2019) 12–24. URL: https://www.sciencedirect.com/science/article/pii/S1389128618305000. doi:https://doi.org/10.1016/j.comnet.2019.01.036.

[20] A. Leivadeas, M. Falkner, I. Lambadaris, G. Kesidis, Optimal virtualized network function allocation for an sdn enabled cloud, Computer Standards & Interfaces 54 (2017) 266–278. URL: https://www.sciencedirect.com/science/article/pii/S0920548917300168. doi:https://doi.org/10.1016/j.csi.2017.01.001, sI: Standardization SDN—&NFV.

[21] M. Huang, W. Liu, T. Wang, A. Liu, S. Zhang, A cloud–mec collaborative task offloading scheme with service orchestration, IEEE Internet of Things Journal 7 (2020) 5792–5805. doi:10.1109/JIOT.2019.2952767.

[22] Z. Yang, R. Gu, H. Li, Y. Ji, Approximately lossless model compression-based multilayer virtual network embedding for edge-cloud collaborative services, IEEE Internet of Things Journal (2023) 1–1. doi:10.1109/JIOT.2023.3259380.

[23] Y. Yue, B. Cheng, X. Liu, M. Wang, B. Li, J. Chen, Resource optimization and delay guarantee virtual network function placement for mapping sfc requests in cloud networks, IEEE Transactions on Network and Service Management 18 (2021) 1508–1523. doi:10.1109/TNSM.2021.3058656.

[24] M. Bunyakitanon, A. P. da Silva, X. Vasilakos, R. Nejabati, D. Simeonidou, Auto-3p: An autonomous vnf performance prediction & placement framework based on machine learning, Computer Networks 181 (2020) 107433.

[25] T. Subramanya, D. Harutyunyan, R. Riggio, Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks, Computer Networks 166 (2020) 106980. URL: https://www.sciencedirect.com/science/article/pii/S1389128619310254. doi:https://doi.org/10.1016/j.comnet.2019.106980.

[26] Z. Yang, R. Gu, Y. Ji, Virtual network function placement based on differentiated weight graph convolutional neural network and maximal weight matching, in: 2021 IEEE Symposium on Computers and Communications (ISCC), 2021, pp. 1–7. doi:10.1109/ISCC53001.2021.9631516.

[27] Cheng et al., Deployment of service function chain for nfv-enabled network with delay constraint, in: ICET, 2018, pp. 383–386.

[28] Soualah et al., Energy efficient algorithm for vnf placement and chaining, in: IEEE/ACM CCGRID, 2017, pp. 579–588.

[29] Assi et al., Energy-aware mapping and scheduling of network flows with deadlines on vnfs, IEEE Transactions on Green Communications and Networking 3 (2019) 192–204.

[30] N. H. Thanh, N. Trung Kien, N. V. Hoa, T. T. Huong, F. Wamser, T. Hossfeld, Energy-aware service function chain embedding in edge–cloud environments for iot applications, IEEE Internet of Things Journal 8 (2021) 13465–13486. doi:10.1109/JIOT.2021.3064986.

[31] R. Moosavi, S. Parsaeefard, M. A. Maddah-Ali, V. Shah-Mansouri, B. H. Khalaj, M. Bennis, Energy efficiency through joint routing and function placement in different modes of sdn/nfv networks, Computer Networks 200 (2021) 108492. URL: https://www.sciencedirect.com/science/article/pii/S1389128621004345. doi:https://doi.org/10.1016/j.comnet.2021.108492.

[32] G. Sun, Y. Li, H. Yu, A. V. Vasilakos, X. Du, M. Guizani, Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks, Future Generation Computer Systems 91 (2019) 347–360. URL: https://www.sciencedirect.com/science/article/pii/S0167739X1831848X. doi:https://doi.org/10.1016/j.future.2018.09.037.

[33] H. U. Adoga, D. P. Pezaros, Network function virtualization and service function chaining frameworks: A comprehensive review of requirements, objectives, implementations, and open research challenges, Future Internet 14 (2022). URL: https://www.mdpi.com/1999-5903/14/2/59. doi:10.3390/fi14020059.

[34] G. MIRJALILY, Z. LUO, Optimal network function virtualization and service function chaining: A survey, Chinese Journal of Electronics 27 (2018) 704–717. doi:10.1049/cje.2018.05.008.

[35] W. Attaoui, E. Sabir, H. Elbiaze, M. Guizani, Vnf and cnf placement in 5g: Recent advances and future trends, IEEE Transactions on Network and Service Management (2023) 1–1. doi:10.1109/TNSM.2023.3264005.

[36] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, X. Fu, Recent advances of resource allocation in network function virtualization, IEEE Transactions on Parallel and Distributed Systems 32 (2021) 295–314. doi:10.1109/TPDS.2020.3017001.

[37] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Technical Report CMU/SEI-90-TR-021, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[38] L. A. Haibeh, M. C. E. Yagoub, A. Jarray, A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches, IEEE Access 10 (2022) 27591–27610. doi:10.1109/ACCESS.2022.3152787.

[39] L. Özbakir, A. Baykasoğlu, P. Tapkan, Bees algorithm for generalized assignment problem, Applied Mathematics and Computation 215 (2010) 3782 – 3795. doi:10.1016/j.amc.2009.11.018.

[40] T. Koch, T. Berthold, J. Pedersen, C. Vanaret, Progress in mathematical programming solvers from 2001 to 2020, EURO Journal on Computational Optimization 10 (2022) 100031. URL: https://www.sciencedirect.com/science/article/pii/S2192440622000077. doi:https://doi.org/10.1016/j.ejco.2022.100031.

[41] D. Machado, A benchmark of optimization solvers for genome-scale metabolic modeling, bioRxiv (2023). doi:10.1101/2023.04.11.536343.

[42] Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

[43] T. Q. Dinh, J. Tang, Q. D. La, T. Q. S. Quek, Offloading in mobile edge computing: Task allocation and computational frequency scaling, IEEE Transactions on Communications 65 (2017) 3571–3584.

[44] Strategy for the promotion of 5g technology, 2022. URL: https://espanadigital.gob.es/sites/agendadigital/files/2022-01/Strategy-for-the-promotion-of-5G.pdf, online; accessed 5 September 2023.

[45] Y. Siriwardhana, P. Porambage, M. Liyanage, M. Ylianttila, A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications, and technical aspects, IEEE Communications Surveys & Tutorials 23 (2021) 1160–1192. doi:10.1109/COMST.2021.3061981.

[46] Opencv's docs, 2023. URL: https://docs.opencv.org/4.x/, online; acc. 20 June 2023.

[47] Docker's docs, 2023. URL: https://docs.docker.com, online; acc. 10 June 2023.

[48] stress-ng tool, 2023. URL: http://manpages.ubuntu.com/manpages/bionic/man1/stress-ng.1.html, online; accessed 10 June 2023.

[49] Hirst et al., Watts up? pro ac power meter for automated energy recording, Behavior Analysis in Practice 6 (2013) 82–95.

[50] iperf tool, 2023. URL: https://iperf.fr, online; acc. 10 June 2023.

[51] Z. Wei-guo, M. Xi-lin, Z. Jin-zhong, Research on kubernetes' resource scheduling scheme, in: Proceedings of the 8th International Conference on Communication and Network Security, ICCNS 2018, Association for Computing Machinery, New York, NY, USA, 2018, p. 144–148. doi:10.1145/3290480.3290507.

[52] P. Lai, Q. He, J. Grundy, F. Chen, M. Abdelrazek, J. G. Hosking, Y. Yang, Cost-effective app user allocation in an edge computing environment, IEEE Transactions on Cloud Computing (2020) 1–1. doi:10.1109/TCC.2020.3001570.

[53] Cuiet et al., Demand response in noma-based mobile edge computing: A two-phase game-theoretical approach, IEEE Transactions on Mobile Computing (2021) 1–1.

[54] S. Tiwari, C. Bhatt, A comprehensive study on cloud computing: Architecture, load balancing, task scheduling and meta-heuristic optimization, in: J. Hemanth, D. Pelusi, J. I.-Z. Chen (Eds.), Intelligent Cyber Physical Systems and Internet of Things, Springer International Publishing, Cham, 2023, pp. 137–162.

[55] T. Chen, R. Bahsoon, X. Yao, A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems, ACM Comput. Surv. 51 (2018). URL: https://doi.org/10.1145/3190507. doi:10.1145/3190507.

[56] D. T. Campbell, J. C. Stanley, Experimental and quasi-experimental designs for research, Ravenio Books, 2015.

[57] Etsi - standards for nfv, https://www.etsi.org/technologies/nfv, 2023. Online; accessed 10 June 2023.

[58] Osm vnf onboarding guide, 2023. URL: https://osm.etsi.org/docs/vnf-onboarding-guidelines/, online; acc. 12 June 2023.

[59] B. Yi, X. Wang, K. Li, S. K. Das, M. Huang, A comprehensive survey of network function virtualization, Comput. Networks 133 (2018) 212–262.

[60] L. Mamushiane, A. A. Lysko, T. Mukute, J. Mwangama, Z. D. Toit, Overview of 9 open-source resource orchestrating etsi mano compliant implementations: A brief survey, in: 2019 IEEE 2nd Wireless Africa Conference (WAC), 2019, pp. 1–7. doi:10.1109/AFRICA.2019.8843421.

[61] A. Reid, A. González, A. Armengol, G. de Blas, M. Xie, P. Grønsund, P. Willis, P. Eardley, F. Salguero, Osm scope, functionality, operation and integration guidelines, ETSI, White Paper (2019).

[62] Opnfv's docs, 2023. URL: https://docs.opnfv.org/, online; acc. 20 June 2023.

[63] Onap's docs, 2023. URL: https://docs.onap.org/, online; acc. 20 June 2023.

[64] Cloudfy docs, 2023. URL: https://docs.cloudify.co/latest/, online; accessed 20 June 2023.

[65] Osm scope, functionality, operation and integration guidelines, 2023. URL: http://osm-download.etsi.org/ftp/Documentation/201902-osm-scope-white-paper, online; accessed 20 June 2023.

[66] Scaphandre documentation, 2023. URL: https://hubblo-org.github.io/scaphandre-documentation/, online; accessed 20 June 2023.