

COSMOS: An Orchestration Framework for Smart Computation Offloading in Edge Clouds

George Papathanail, Ioakeim Fotoglou, Christos Demertzis, Angelos Pentelas,
Kyriakos Sgouromitis, Panagiotis Papadimitriou

University of Macedonia, Greece

{papathanail, fotoglou, demertzis, apentelas, sgouromitis, papadimitriou}@uom.edu.gr

Dimitrios Spatharakis, Ioannis Dimolitsas, Dimitrios Dechouniotis, Symeon Papavassiliou

National Technical University of Athens, Greece

{dsatharakis@netmode, jdimol@netmode, ddechou@netmode, papavas@mail}.ntua.gr

Abstract—The evolution of Internet of Things (IoT) has sparked significant research interest in edge computing. Within this scope and given the ever-increasing number of IoT and mobile devices, computation offloading is emerging as a cutting-edge and significant research area with enormous potential and practical applications.

In this respect, we present the architecture design and experimental evaluation of an orchestration framework for smart computation offloading from IoT or mobile devices to edge cloud servers. The proposed orchestration platform, namely COSMOS, includes control-plane components for workload prediction, load balancing, and admission control. COSMOS is particularly tailored to the needs of an object identification service that receives images from a multitude of Points of Interest (PoIs), performs object identification using a trained model (based on Tensorflow), calculates the prediction accuracy, and finally returns to the end-users the identification outcome and accuracy along with useful information about the identified object. COSMOS has been deployed and evaluated in a large-scale experimental facility that employs OpenStack and OpenSourceMANO (OSM) for Network Function Virtualization (NFV) orchestration. Our experimental results indicate the feasibility of computation offloading for this object identification service and further uncover useful insights in terms of performance and scalability.

I. INTRODUCTION

In the dawn of the 5G era, the demand for computational resources has been increasing at a significant and accelerating rate. Novel emerging applications, such as Augmented Reality (AR) [1], object recognition, dynamic interactions between devices (*e.g.* autonomous vehicles), have been a major driving factor for the advent and evolution of multi-access edge computing (MEC) [2]. In essence, MEC harnesses the computational power at the network edge (*i.e.*, servers or other computing resources close to base-stations, Wi-Fi access points, or the backhaul). MEC is a key component for the realization of Internet of Things (IoT) and mobile applications in smart city environments [3]. In particular, MEC provides several benefits for such applications: (i) access to powerful computational assets, (ii) reduction of the energy consumption in IoT/mobile devices, (iii) lower network latency compared to core clouds, and (iv) higher performance of

application components deployed in an edge cloud (compared to their local execution in client devices). In this respect, the success of MEC depends significantly on the ability to offload computationally-intensive tasks from mobile or IoT devices to proximate edge clouds [4]. The offloading decision can be based on several network, application, and mobile device parameters (*e.g.*, battery level), using existing methods [5].

Besides application task offloading, the remote execution of computational tasks in MEC servers entails significant requirements in terms of resource and service orchestration. Such requirements include, for instance, the ability to perform admission control on incoming task offloading requests, the assignment of requests to the edge cloud with respect to MEC provider allocation policies (*e.g.*, load balancing), as well as the prediction of tasks' computational demands in order to prevent over- or under- provisioning of CPU resources in MEC servers.

In this respect, we present an orchestration framework for smart computation offloading in edge clouds. The proposed orchestration system, called COSMOS, has been particularly designed to facilitate the execution of an object identification service for mobile users. In this particular scenario, the visitors of a touristic area use their camera-equipped devices to take snapshots or short-duration videos of a Point of Interest (PoI) (*e.g.*, museum, square, statue) in order to receive useful information via an object recognition service. Since PoI identification is a computationally intensive task, certain instances of the identification service are deployed in a proximate edge cloud in order to alleviate the computational burden from the mobile device and also reduce its power consumption.

The proposed orchestration framework has been designed with the following objectives in mind: (i) the reduction of response time in order to avoid any user experience impairments, due to the computational offload, (ii) the admission control of offloading requests, based on the residual computational capacity of the edge cloud, (iii) the dynamic load balancing of the object identification instances among the allocated MEC servers.

COSMOS has been deployed and evaluated in a large-scale experimental facility of the 5GinFIRE project [6], using OpenSourceMANO (OSM) for Network Function Virtualization (NFV) orchestration [7], [8]. The experimental evaluation of COSMOS is conducted on a realistic environment with various PoIs located in a square, part of the facility. The tests are executed by real mobile users, connected via Wi-Fi access points to the facility's edge cloud infrastructure. The measured response times validate the efficiency of COSMOS orchestration functions, such as load balancing and admission control.

The remainder of this paper is organized as follows. Section II provides an overview of the enabling technologies for the proposed orchestration system. In Section III, we present the COSMOS architecture design and the functionality of its main components. In Section IV, we discuss in detail our experimental results and assess the performance and scalability of COSMOS in the context of object identification. Section V discusses related work. Finally, in Section VI, we highlight our conclusions.

II. ENABLING TECHNOLOGIES

In this section, we provide an overview of the technologies employed for the deployment of the proposed orchestration framework.

A. OpenStack

OpenStack [9] is an open-source software for Infrastructure as a Service (IaaS), structured by a stack of software tools used to build and manage cloud computing platforms for public and private clouds. This is achieved through the interoperability of the following core OpenStack modules: the compute module, namely *Nova*, is responsible for cloud server resource allocation and management; *Neutron* implements the required network configurations (e.g., routing, tunneling, virtual queuing, QoS); *Glance* acts in synergy as a database, which maintains all the compatible server images provided to OpenStack; *Keystone* handles user authentication and authorization; *Horizon* is a web-based GUI that enables cloud management through an intuitive dashboard. In NFV infrastructures and orchestration platforms [10], OpenStack commonly serves as a Virtualized Infrastructure Manager (VIM), interfacing with the NFV orchestrator (e.g., OSM), which we briefly discuss below.

B. OpenSourceMANO

OSM [11] is a widely-used NFV orchestration platform, whose operation and functionality is governed by the NFV Management and Orchestration (MANO) reference architecture, as defined by ETSI NFV. The architecture of OSM consists of the *LifeCycle Manager (LCM)*, the *Resource Orchestrator (RO)*, and the *VNF Configuration and Abstraction Module (VCA)*. LCM constitutes the basic orchestrator for network services. LCM interacts with RO, which is responsible for provisioning network services over a connected VIM (e.g., Openstack). RO is in charge of the resource lifecycle at the

VIM layer, enabling the VIM to perform resource allocation, according to the requirements specified in the VNF descriptors. VCA acts as a generic virtualized network function (VNF) manager, relying on Juju. VCA primarily handles the interaction with VNFs and, more specifically, is used for the so-called *Day-2* configurations.

C. TensorFlow

TensorFlow [12] is a Google-powered open-source machine learning platform, providing a comprehensive and flexible ecosystem of tools and libraries for simplified model building using intuitive high-level APIs, such as Keras [13]. Tensorflow is used by COSMOS for the identification of PoIs (e.g., monuments, landmarks, exhibits) in touristic areas. The object identification application consists of three phases; namely, (i) *data pre-processing*, (ii) *model building* and (iii) *model training and evaluation*. The model receives a multi-dimensional array as input, which is otherwise known as a tensor (i.e., a n-dimensional vector or matrix capable of representing all types of data) and by constructing an operations graph, the tensor flows through the list of these operations to determine the output. TensorFlow has been packaged as a VNF, enabling its onboarding into NFV orchestrator platforms, such as OSM.

III. ORCHESTRATION ARCHITECTURE

In this section, we present the design of the COSMOS orchestration architecture and the functionality of its main components for mobile computation offloading in edge clouds. The particular architecture is designed to meet the requirements of PoI identification in touristic areas; however, its applicability is broader, since most of its components (e.g., workload prediction, load balancing, admission control) can be used for MEC resource orchestration with alternative processing workloads.

A. Architecture Overview

The proposed orchestration system relies a centralized controller (deployed in the edge cloud), which receives requests from mobile users (e.g., visitors in a touristic area) for the offloading of object identification into the MEC, as shown in Fig. 1. The COSMOS orchestration system is designed to handle concurrently requests from multiple users, which may be potentially connected to different Wi-Fi access points.

The COSMOS controller consists of the following main components: workload prediction, load balancing, admission control, and monitoring. Workload prediction employs Kalman filtering [14] to estimate the number of mobile user requests and augment the operation of admission control, which rejects requests that cannot be handled by the deployed object identification instances (implemented using TensorFlow) in the edge cloud. In turn, the load balancing module assigns the incoming requests among the deployed TensorFlow instances, taking into account the number of requests and the processing load of each instance. We discuss in further detail the operation of the admission control and load balancing in Sections III-B and III-C, respectively.

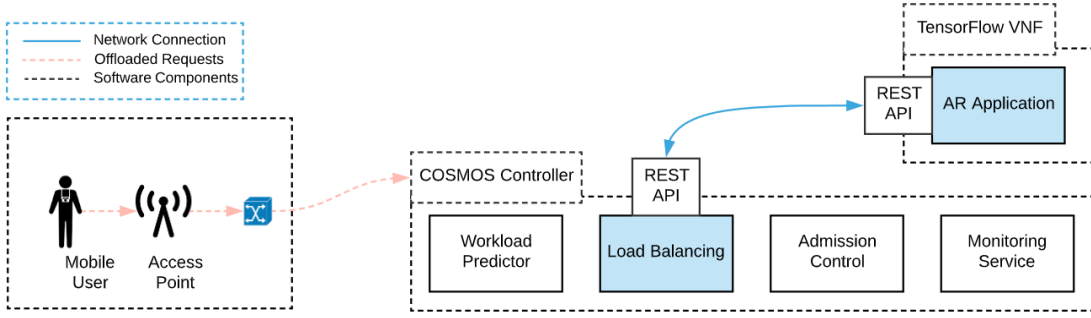


Fig. 1: COSMOS orchestration architecture.

Both the controller and the TensorFlow instances are deployed as separate VNFs (Fig. 2), which are managed and orchestrated by Openstack and OSM, within a large-scale experimental infrastructure (*i.e.*, Bristol Smart City testbed [15]). Although all control plane elements are currently co-located in the same VNF, alternative deployment models (*i.e.*, control plane modules deployed in separate VNFs) are also possible. The TensorFlow VNF instances communicate with the controller via a REST-API, using internal connections. It is worth noting that although the proposed architecture has been specified with a single MEC domain in mind, it can be expanded upon for federated cloud infrastructures.

B. Admission Control

In order to sustain high Quality of Experience (QoE) for the mobile users, the controller admits incoming requests based on the level estimated by Kalman filtering [14]. Requests exceeding this level are rejected, along with requests initiated from user devices with signal strength below a fixed threshold. This control process acts, in essence, as a form of admission control. To this end, a database, residing within the controller, maintains monitoring data for the requests and the processing load of the TensorFlow VNF instances.

In order to accommodate the incoming traffic while reducing the response time of object identification, an additional admission control mechanism has been implemented. More precisely, the users send an initial request to inform the controller of their position and current signal strength. Subsequently, the users are notified whether their requests are accepted, depending on a predefined threshold of signal strength that is correlated with their position. In case of request rejection, the object identification is executed locally on the user's device.

C. Load Balancing

Upon the execution of admission control, the offloaded traffic generated by the mobile users should be distributed among the running TensorFlow instances that carry out the object identification. To better accommodate this control process, time in our framework is quantized in discrete time intervals. Upon each time interval, while actively monitoring all the incoming requests, the controller computes the distribution of the incoming workload for the following time interval.

This load balancing process is performed in an on-line and proactive manner, while invoking an internal prediction mechanism, *i.e.*, the *Workload Predictor*. The latter provides an estimation of the expected number of requests within the time interval (based on Kalman Filter [14]). Furthermore, the load balancer takes into account the processing load of each running TensorFlow instance, with the aim of directing requests to the instances with the lower utilization level.

IV. EVALUATION

This section provides a detailed description of our experimental environment (Section IV-A), the employed TensorFlow model (Section IV-B), and the evaluation results (Section IV-C) of the COSMOS orchestration framework.

A. Experiment Setup

According to the description of the COSMOS orchestration framework (Section III), a service chain [16] consisting of four VNFs is specified and deployed in the experimental facility. The controller VNF is responsible for admission control and load balancing among the remaining three TensorFlow VNFs. The controller VNF is associated with 4 CPU cores, 8GB of RAM and 20GB of storage space. For the TensorFlow VNFs, we use three different flavors, whose resource specifications are shown in Table I. As explained in Section III-C, accepted

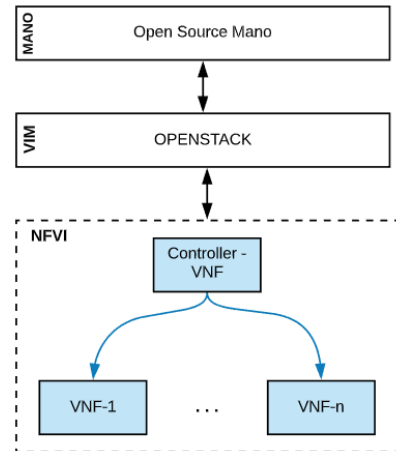


Fig. 2: Experimental deployment of COSMOS.

TABLE I: TensorFlow flavor specifications

Flavors	vCPUs (cores)	RAM (GB)	Storage (GB)
Small	2	2	10
Medium	4	4	20
Large	8	8	20

user requests are distributed among these Tensorflow flavors with the aim of load balancing.

For our experimentation, we create the appropriate software images, which correspond to the functions required for our evaluation. The functions of the controller, alongside with its corresponding API, are installed in the Linux-based Ubuntu Bionic 18.04 LTS operating system as the first image, while the second image features the TensorFlow-based object recognition functions. These VNFs are deployed, managed, and orchestrated based on the MANO principles, using OpenStack and OSM. In this respect, appropriate VNF and Network Service (NS) descriptors have been defined for the description of all required functions and their chaining as a VNF-graph.

B. TensorFlow Model

The open-source TensorFlow object detection framework facilitates the construction, training and deployment of the object detection model. The object detection API, which is used for object identification during the experiment, provides the capability of running inference jobs on pre-trained object detection models; thus, the latter are not required to be trained from scratch. Regarding the model selection, the faster RCNN Inception V2 COCO model is employed, due to its high speed predictions (58ms) and decent mean Average Precision (28 mAP). Specifically, this model is selected, as it was pre-trained on the Common Objects in Context (COCO) [17] dataset. COCO is a large-scale object detection, segmentation and captioning dataset, which consists, among others, of 330K images (>200 labelled), 1.5 million object instances, 80 object categories and 250K people with key points.

As for the model data, it is stored in the form of weights, which are used as initial weights in the pre-trained model, and are consequently readjusted after the addition of our own dataset and completion of the re-training. Our training data consists of approximately 5 minutes of high resolution (1920x1080), 30 frames/second video (1 minute per object), captured from the Millennium square of Bristol Smart City testbed. To extract the images, we use the FFmpeg software [18] to create 100 images (with resolution of 800x600) per class, resulting in a total of 500 pictures. Ideally, images should differ significantly from each other, allowing for a greater variety of angles and lighting conditions. Consequently, we create a mirror transformation of these 500 pictures, ending up in a combined total of 1000 pictures.

C. Experimental Results

Our experimental evaluation aims at assessing the feasibility of object identification offloading into an edge cloud

infrastructure. To this end, we use the metric of total response time, which is further subdivided into individual constituent components, *i.e.*, computation and transmission time. As such, we enable the offloading of the most demanding part of the identification on the mobile edge cloud infrastructure with the goal of utilizing the substantially superior hardware of the edge servers, compared to the consumer-grade hardware of the client devices. For our experimentation, we have generated a dataset consisting of 15224 object detection requests, which are triggered within thirty-second time intervals. Ultimately, 653 such time intervals are recorded, each one defined by the number of requests that occurred. These 653 intervals are classified into records, depending on the number of requests, while averaging the corresponding values.

According to Fig. 3, the application's execution, *i.e.*, response time, steadily remains under 6.3 seconds, while the minimum time observed is 3.2 seconds. The most significant insight that can be drawn from this plot, though, is the non-exponential nature of the increase in the average response, computation, and transmission times. This can be attributed to the efficient utilization of the three distinct flavors that we have deployed, as explained in Section III. In particular, the selection of the appropriate flavor is determined by the volume of incoming requests. Thus, a near-linear increase in the key metrics is deemed to be satisfactory. However, we observe a significant variation in the standard deviation, which is discussed in the following.

We further observe a strong correlation between response (Fig. 3) and computation time (Fig. 4), not only in the distribution of the observations, but also in the values of each individual measurement. Considering the response time as a metric that essentially translates into a combination of computation and transmission times, we can infer that computation time is the dominant factor of the overall application execution time. As for the fluctuations on the observable values, we reasonably attribute these to the varying complexity of the calculations, depending on each individual image sent to the application.

Accordingly, the scale of the average transmission time is an order of magnitude lower (Fig. 5). Hence, the contribution of transmission time to the overall response time for the application is so minimal that borders on insignificance, with computation time commanding by far the largest influence on the total response time. Additionally, some further insights that can be gained from Fig. 5 is a low deviation from the mean for all transmissions due to the admission control mechanism, and a negligible, near-linear, increase when the number of requests increases.

To further analyze the application's behavior in terms of response time, we divide our observations into two separate groups. This separation is driven from the median of our dataset, *i.e.*, the value that segregates records in two. The first half is comprised of time intervals where less than the *median* number of requests are triggered, while the second includes the ones exceeding it. Fig. 6 illustrates the correspondence of the two variables' distributions, with the second group

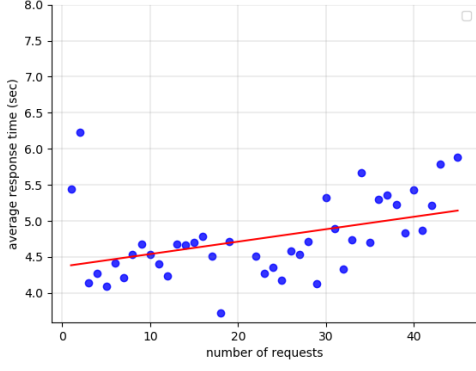


Fig. 3: Average response time (grouped observations).

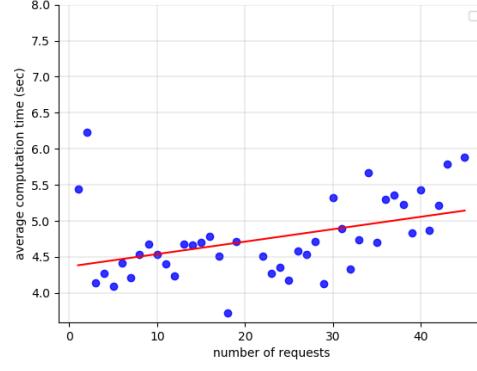


Fig. 4: Average computation time (grouped observations).

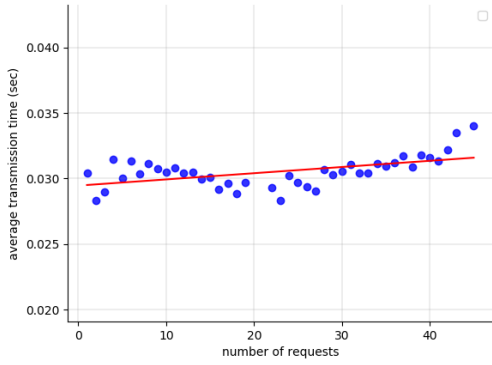


Fig. 5: Average transmission time (grouped observations).

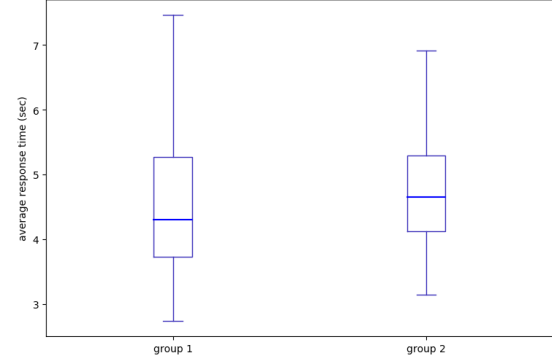


Fig. 6: Boxplot of response time (split into groups by the median of requests).

demonstrating a slightly narrower deviation. We point out that the effectiveness of utilizing a range of different flavors becomes apparent. That is, the mean of the second group, *i.e.*, the group handling a larger number of requests, does not substantially differ from the one of the first group.

Drawing from the discussion above, we emphasize on three key points. The primary driver of the response time is the delay incurred for processing, a task which is accelerated by its offloading to the MEC servers. Furthermore, the transmission time is negligible with respect to the total response time. Finally, load balancing is considered of critical importance for the efficient utilization of MEC resources, since only an average 7.96% of the total number of requests is rejected by the COSMOS controller and, thus, executed locally on the client device.

V. RELATED WORK

This section provides an overview of related work on the domains of (i) computational offloading to edge clouds, and (ii) NFV orchestration.

Edge Computing: An important objective of MEC is computation offloading in order to satisfy requirements, such as the reduction of power consumption in mobile devices and

low response time from mobile and IoT applications. In this respect, Avgeris et al. [4] present an offloading framework for IoT-enabled applications, which is based on vertical and horizontal scaling of the MEC infrastructure. The scaling approach is based on linear systems combined with set-theoretic controllers. In [19], a scalable edge computing framework for early fire detection is presented. An image processing service is hosted on an edge cloud and the regulation of the application's response time is based on linear optimization and state feedback controllers. Leivadreas et al. [20] propose a meta-heuristic approach for the placement and the deployment of the VNFs of IoT-enabled applications in order to minimize the end-to-end communication delay and the overall deployment cost. MAGA [21] is a mobility-aware, genetic algorithm-based offloading and task allocation system that seeks to reduce the energy consumption of the mobile devices and meet the response time requirements of IoT applications.

Beyond existing solutions, our work provides a holistic resource orchestration framework for mobile and IoT-enabled applications. Load balancing is carried out based on the processing capabilities of the predefined VNF flavours. Furthermore, the offloading decision is based on contextual information (*i.e.*, user's position and wireless signal strength)

and the proposed workload estimation methodology, which guarantees the response time requirements and the efficient allocation of MEC resources.

NFV Orchestration. A significant amount of work has been conducted on establishing and advancing the NFV paradigm [22]. Most work on NFV has focused on NFV orchestration, which encompasses aspects, such as service chain embedding [16], VNF scaling [23], and service chaining.

Besides OSM [11], another well-known orchestration platform, which has been developed in the context of the T-NOVA project, is TeNOR [10]. TeNOR acts as an NFV orchestration module and is responsible for the provisioning, configuration, monitoring, and optimized operation of service chains over virtualized infrastructures. However, neither TeNOR nor similar NFV orchestration mechanisms (*e.g.*, [8], [24]) explicitly address the various needs of computational offloading orchestration in edge clouds. Instead, they are mainly targeted at core cloud environments which, inherently, are less dynamic and agile than edge cloud infrastructures.

COSMOS overcomes the limitations of such orchestration mechanisms, offering mobility-aware computation offloading for mobile and IoT-enabled applications. In particular, COSMOS focuses on dynamic aspects of NFV orchestration (*e.g.*, dynamic load balancing), going beyond static service provisioning which is the main focus of most existing NFV orchestration mechanisms. Finally, the deployment of COSMOS allows for a preliminary examination of the extent to which existing NFV orchestration platforms (*e.g.*, OSM) can cope with service deployment and scaling for mobile and IoT applications in smart city environments.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we discussed the architecture and experimental evaluation of an orchestration framework for computation offloading from IoT/mobile devices to edge clouds. COSMOS effectively couples edge computing with artificial intelligence and software-based network principles, such as NFV and SDN. Our evaluation results from a large-scale experimental facility show that mobile computational offloading constitutes a feasible approach for alleviating the computational burden from client devices, especially for computationally-intensive workloads such as object identification. The utilization of proximate MEC resources further facilitates the scaling of such services in response to evolving demands from mobile users. At the same time, the proposed orchestration framework sustains high user-perceived service quality, as quantified in terms of response time.

Our future work will pursue certain improvements in the orchestration platform. More specifically, we aim at exploiting the scaling capabilities of the latest OSM releases for the implementation of an efficient auto-scaling mechanism. Furthermore, we plan to employ a precise indoor positioning and workload prediction method in order to enable context-aware offloading decisions.

REFERENCES

- [1] W. Zhang, B. Han, and P. Hui, "On the networking challenges of mobile augmented reality," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 2017, pp. 24–29.
- [2] "ETSI Multi-access edge computing," <http://www.etsi.org/technologies/multi-access-edge-computing>.
- [3] N. Kalatzis, M. Avgeris, D. Dechouniotis, K. Papadakis-Vlachopapadopoulos, I. Roussaki, and S. Papavassiliou, "Edge computing in iot ecosystems for uav-enabled early fire detection," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2018, pp. 106–114.
- [4] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading: A control-theoretic approach," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, p. 23, 2019.
- [5] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
- [6] "5GinFIRE project," <http://5ginfire.eu>.
- [7] "ETSI Network Function Virtualization," <http://www.etsi.org/technologies-clusters/technologies/nfv>.
- [8] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: a framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015, pp. 121–136.
- [9] "Openstack," <http://www.openstack.org>.
- [10] M.-A. Kourtis, M. J. McGrath, G. Gardikis, G. Xilouris, V. Riccobene, P. Papadimitriou, E. Trouva, F. Liberati, M. Trubian, J. Batallé *et al.*, "T-nova: An open-source mano stack for nfv infrastructures," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 586–602, 2017.
- [11] "OSM," <http://osm.etsi.org>.
- [12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems, 2015," *Software available from tensorflow.org*, vol. 1, no. 2, 2015.
- [13] "KERAS," <http://github.com/charlespwd/project-title>.
- [14] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.
- [15] "Bristol 5G Testbed," <https://5ginfire.eu/university-of-bristol-5g-testbed>.
- [16] D. Dietrich, A. Abujoda, A. Rizk, and P. Papadimitriou, "Multi-provider service chain embedding with nestor," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 91–105, 2017.
- [17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.
- [18] F. Developers, "Ffmpeg," 2016.
- [19] M. Avgeris, D. Spatharakis, D. Dechouniotis, N. Kalatzis, I. Roussaki, and S. Papavassiliou, "Where there is fire there is smoke: a scalable edge computing framework for early fire detection," *Sensors*, vol. 19, no. 3, p. 639, 2019.
- [20] A. Leivadadas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "Vnf placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, p. 69, 2019.
- [21] Y. Shi, S. Chen, and X. Xu, "Maga: A mobility-aware computation offloading decision for distributed mobile cloud computing," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 164–174, 2017.
- [22] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2015.
- [23] Z. Cao, A. Abujoda, and P. Papadimitriou, "Distributed data deluge (d3): Efficient state management for virtualized network functions," in *2016 IEEE Conference on Computer Communications Workshops*, April 2016, pp. 782–787.
- [24] M. Mechtri, C. Ghribi, O. Soualah, and D. Zeghlache, "Nfv orchestration framework addressing sfc challenges," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 16–23, 2017.