End-to-end Fiber & 5G

A Measurement-Based Model of MEC in the 5G System

by

Riccardo Fedrizzi, Cristina Emilia Costa, and Fabrizio Granelli

Abstract

Multi-access edge computing (MEC) represents an emerging solution to address the issues related to reliability, availability, context awareness and low latency in modern and future mobile networks. Despite the great interest and effort on developing MEC solutions, few works are available in the literature to model the existing trade-offs between available resources and related performance of MEC nodes. This paper aims to provide a methodology to fill this gap by proposing a measurement-based approach to derive an effective model for MEC nodes, capable of capturing the existing trade-offs among different performance parameters. A network emulator is used to generate the data to define the model. The paper provides an overview of the different tradeoffs and an analysis of the performance of a single MEC node under CPU and networking limitations.

I. Introduction

Multi-Access Edge Computing, commonly referred to as MEC, is acknowledged as a fundamental component of next-generation mobile networks. It introduces computation capabilities at the edge of the wireless access network, enabling new and challenging Use Cases. One of the most interesting benefits of introducing edge nodes in the network fabric is related to their capability of supporting low latency, high reliability, and context awareness, thus improving the effectiveness of the deployment and improving the performance of modern and future services [1].

The term MEC identifies also the standardized, open environment defined by the related ETSI Industry Specification Group (ISG). ETSI defined a framework for application developers and content providers, the MEC enablement platform [ref], in which the Edge node offers cloud-computing capabilities and an IT services deployment environment. The edge node hosts a virtualization platform where applications may be deployed as containers or VMs. Deployments are dynamic and are orchestrated by a central entity as needed. Mechanisms for traffic steering of the data flow towards the Edge are envisioned but details are not defined in the ETSI standard. The ongoing efforts are maintained harmonized with 3GPP standardization work in a joint effort between the two SDOs. In 5G the steering functionality is commonly implemented by an instance of the User Plane Function (UPF) deployed at the Edge. The UPF is the Network Function responsible in the 5G Core for selectively breaking the GTP tunnel and steering the data flow, in this case, to the Edge Data Network (DN). The UPF itself can be deployed as a Virtual Network Function in the Edge Cloud.

Edge nodes can be geographically distributed at different density degrees. Deployment architectures may include networks in various data center locations, as well as smaller and more pervasive nodes that may be subject to severe constraints in terms of costs and computational capability. It is realistic to think that services providers would face in the future a quite heterogeneous landscape of multi-access edge computing (MEC) nodes (depending on how ”deep” or dense is the Edge).

In the last years, several initiatives and projects focused on developing proofs-of-concept in line with ETSI standards definitions have been carried on[1]. However, better understanding and modeling the edge nodes’ behavior is required in order to enable proper management and optimization of service deployment. This would require a step further with respect to the current state-of-the-art.

Indeed, when we consider the heterogeneity, dynamicity, and constraints of the cloud edge continuum, it is evident that a ”one size fits all” approach is not feasible and that finding the better deployment cannot be a trivial task. The MEC performance depends on the availability of computing resources, which are often limited depending on how,

[1] https://mecwiki.etsi.org/index.php?title=MEC Ecosystem

and which type of, nodes are deployed. For MEC systems to be sustainable and manageable, service providers will need to adopt suitable models that would allow them to dimension and provision their MEC nodes on the network fabric and to understand how to enable them to support the applications and the related KPIs. In this framework, orchestration of the network functions and services plays a central role in the optimization of the available resources.

In the literature, several works propose various orchestration techniques which are based on theoretical models for Edge Nodes behavior. Some previous works provide good preliminary approaches to compare different orchestration techniques in constrained cloud-edge scenarios. Harutyunyan et al. [2] leverage simple delay models to characterise the VNF processing delay which is then used to allocate CPU resources by imposing end-to-end (E2E) QoS constraint on the considered services in the scenario. More complex models are considered for the same objective in [3] and [4] using respectively M/M/1 and M/D/1 queuing models. However, those models, while enabling interesting preliminary evaluations, are not precise enough in modeling the behavior of real edge nodes running real services. To fill this gap, some authors started considering introducing experimental measurements in their models. A good example can be found in [5], where authors conducted experiments in a real setup and collected data useful to derive a model of the virtualization overhead. This model was then adopted for defining a VNF placement algorithm. The drawback of this approach, however, was its high dependency on the virtualization technology used during experimentation. In [6] the authors leverage experimental results to derive a theoretical model relating the CPU usage, network load and packet delay and edge node performance. In this case, the resulting model is tailored to a specific face recognition scenario. The coexistence and parallel execution of the face recognition application in conjunction with other heterogeneous services were not considered as well as the impact of user plane (UP) function behaviors. Both these works give an idea of the behavior of an application in a constrained scenario, but the findings are context-specific and cannot be easily generalized. In [7], the authors introduce a simulator of the Simu5G 5G Core simulator combined with physical edge hosts to evaluate and experiment MEC technologies. In this approach, the E2E 5G network is simulated by design and does not allow to emulate a MEC, which includes the virtualised functions for the UP termination - the user plane function (UPF). In [8], the authors propose OpenLEON as an E2E emulator spanning from mobile users to the edge data center. This is, to the best of our knowledge, one of the closest works to the one presented in this paper. The authors leverage srsLTE and Containernet to design a realistic radio access implementation with a virtualised environment emulating a 3-tier edge data center hosting the core network and services. Various applications are tested, with the main focus to analyse network performance in various LTE channel configurations.

The goal of this paper is to enable the definition of a measurement-based model for studying MEC and MEC-supported application performance and existing tradeoffs. To this aim, we define a measurement campaign by using a network emulator, capable of controlling the amount of resources allocated for the MEC nodes. Based on such measurements, it is possible to define different "acceptable" operating regions outlining the existing trade-offs, e.g. between computing and network performance. In contrast to [8], our work focuses on the edge side, and it aims to study the deployment of several isolated and resource-constrained MEC hosts and to analyze the corresponding performance tradeoffs. For this reason, we selected a network emulation environment, ComNetsEmu [9], which allows deploying containerised applications on top of emulated physical hosts through a Docker-in-Docker approach. This provides more flexibility and allows to effectively emulate MEC hosts with limited computing resources that are shared between the hosted virtualised applications and functions by using Docker provisioning interfaces.

The code developed to deploy the emulation environment and to study MEC performance described in this paper is freely available online[2] to facilitate the reproduction of the achieved results and to support further research activities on this subject.

[2] https://github.com/RiccardoFedrizzi/networking letter

II. The Proposed Methodology

To study the performance of the applications deployed at the Edge and enabled by the MEC system, we designed a 5G-enabled resource-constrained MEC deployment on an emulation environment based on Open Source platforms.

Comnetsemu is seletected as the simulation platform, being capable of extending Mininet and Containernet with a nested virtualization approach based on DockerHosts. On top of the Comnetsemu network emulator UERANSIM and Open5GS are deployed to implement the 5G System, plus some software utilities to collect data and generate variable traffic and CPU load. The resulting architecture of the resulting emulator environment is represented in Figure 1(a). ComNetsEmu enables to deploy an SDN-enabled transport network connecting various hosts emulating the main components of the 5G network: the radio access network, the 5G control plane (CP), and the MECs. The scenario deployed for the performance evaluation is explained in this section and detailed in Figure 1(b).

Proper tools have been implemented to control the scenario and to easily deploy the network, configure the radio access network (RAN) and the 5G core (5GC), and apply the desired applications (APPs) behaviors. Those tools implement functions to collect the status of the various network components (e.g.: container CPU status through the Docker stats exposed through the Docker Client APIs). In parallel to that, APP tools have been implemented to emulate services running on the MECs and consumed by APP clients of the user equipments (UEs). Collection of the measurements is performed using the Publish/Subscribe (Pub/Sub) infrastructure provided by Redis3. In the remainder of this section, the building blocks of the system and their functionalities are described into more detail.

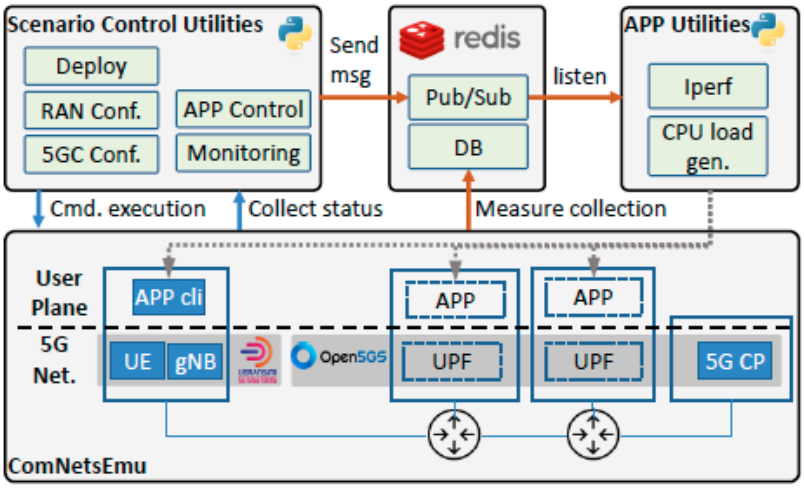

Fig. 1 (a) High-level architecture of the emulation environment.

II.A. Emulated Network Deployment

A proper deployment of the 5G System should be supported by a proper environment integrating both software defined network (SDN) and network function virtualization (NFV). Indeed, Comnetsemu environment natively supports both SDN and NFV through a docker−in−docker framework. All the network functions and applications are deployed within DockerHosts as resource isolated hosts. In Figure 1(a) they are shown in solid-line boxes.Within a DockerHost, APPs and network functions can be emulated by either running processes directly, or by deploying APPContainers exploiting the docker−in−docker concept - see blue boxes and dashed line boxes in Figure 1(a), respectively. APPContainers add a second level of virtualization and allow to emulate containerized applications and network functions running on a resource limited host.

Since by default Docker does not apply CPU limitations, we used Docker built-in functions to limit CPU utilization for specific MEC Docker containers. To control the CPU allocation we set two parameters: the CPU period, CPUperiod, and the CPU quota, CPUquota. CPU quota specifies how much CPU time (in microseconds) the container can use, per CPU period. After a container consumes all its CPU quota, it is throttled for the remainder of the CPU period.

To ease this process we implemented a utility function whereby the CPU constraints of MEC can be set, which uses the cpu perc parameter to set the maximum percentage of the overall system CPU that can be used by all the APPs and the UPF deployed in the specified MEC.

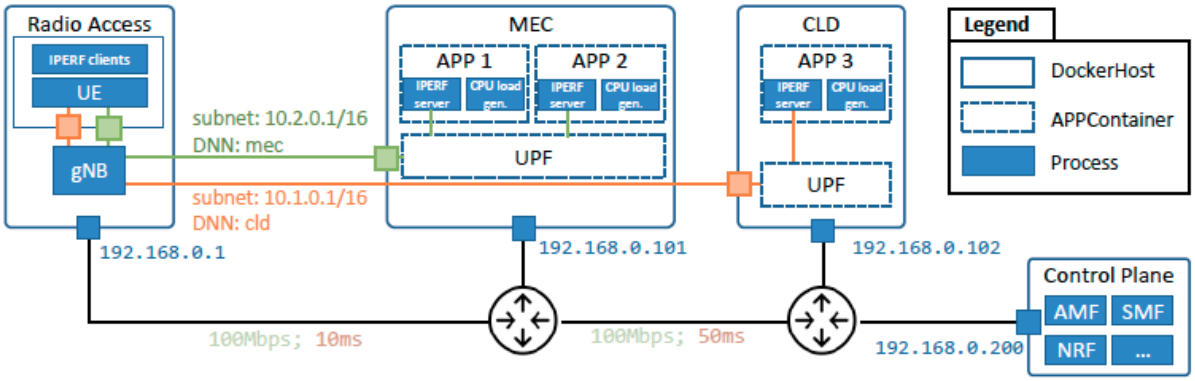

Fig. 1 (b) Emulation components and their integration.

II.B. 5G Core Network

The Open Source implementation from the Open5GS project[3] is used to deploy the 5GC. This solution allows separating the CP and the UP functions, thus supporting the need to have a Data Network in each MEC to reach the APPs. To the best of knowledge of the authors, such implementation of the 5GC allows to emulate a realistic MEC environment with a proper, while avoiding monolithic core network deployment as in [8].

During the deployment phase, all the CP functions are started as processes inside a DockerHost, for simplicity. Their configuration is done through YAML files which are updated on the fly based on the desired scenario. In particular, the session management function (SMF) configuration specifies where to reach the UPFs, their Data Network Name (DNN) and the associated sub-network. Afterwards, an APPContainer running the UPF is deployed in each MEC node and configured to associate a new TUN interface, created on the fly and usually called ogstun, with a DNN and its subnetwork.

Two different logical networks are created to allow reaching the APPs, depending on whether the service is provided through the MEC platform (MEC in Figure 1(b)) or through the cloud (CLD in Figure 1(b)). The corresponding two networks are displayed in green and orange, respectively, in Figure 1(b). Then, the CP needs to be configured with the subscribers' information (International Mobile Subscriber Identity - IMSI, subscribed services, etc.) to allow a new UEs to connect.

II.C. 5G Radio Access

UERANSIM[4], an open source software simulator for the 5G UE and RAN (gNodeB (gNB)), is used to simulate the radio access section of the 5G System. UERANSIM implements the main Radio Resource Control (RRC) procedures and the GTP protocol for the user plane. It allows easing the deployment of an E2E 5G network in a self-contained emulation environment.

Being a simulator, UERANSIM does not implement the protocol stack below the RRC, preventing the emulation of scenarios in which lower layers are needed - however this aspect is beyond the scope of this paper. To avoid such limitation, it would be necessary either to use a physical RAN implementation (e.g., srsRAN), which requires rather costly external hardware and complicates the scenario definition, or more advanced network simulators (e.g., NS3), which typically do not support the required accuracy that an emulation environment might provide in terms of a realistic implementation of the UPF in the MEC and the corresponding workload. Both options

[3] https://open5gs.org
[4] https://github.com/aligungr/UERANSIM

are possible in the Comnetsemu environment, but the authors prefer to focus on a fully integrated and consistent scenario capable of running on a single common PC platform, as previously described.
During the deployment phase, the gNB is deployed as a process in the RAN DockerHost. Once the gNB process is started, it establishes an NG Application Protocol (NGAP) connection with the access & mobility management function (AMF) network function in the 5GC. To add and connect new UEs in the scenario can be easily automated thanks to dedicated control tools that have been implemented by the authors, which starts an UE and connects it by specifying a DNN, used to discriminate between MECs, and a slice type, used to specify the corresponding QoS parameters. Every time a UE connection is started, a new TUN interface is created which allows to use of the 5G connection and reach the MEC APPs as shown in Figure 1(b). The IP address of the TUN interface is assigned by the SMF within the range of the DNN sub-network.
Each UE can have several active connections, with different DNN and slice types. The IP addresses of an UE can be retrieved based on the triplet [ID,DNN, SST].

II.D. Emulated Applications

With the purpose to maintain generality while enabling a proper analysis of the MEC performance, we decided to classify applications into three categories: communication intensive applications, communication intensive applications and mixed applications.
Indeed, most MEC applications expected to fall either in the communication intensive category (i.e. requiring mainly a given throughput level, or offering a quality of service directly related to the achievable throughput level), communication intensive (i.e. requiring mainly CPU resources and performing mostly computation), or something in the middle (i.e. having multiple mixed KPIs). Indeed, the UPF used to steer the traffic to the MEC platform can be hosted and use resources from the platform itself, thus representing a clear example of a communication intensive service. To emulate the above application profiles, control utility functions have been implemented in the emulator in order to (i) control the data traffic between an UE and an APP, and (ii) emulate the APP computation by generating CPU load in its APPContainer.
In the next sections and overall in this paper, we will assume that each MEC node resources should be shared among all the applications and functionalities running in such node (i.e. including the UPF and other ”core” functionalities). The authors believe that this represents a more realistic situation of MEC deployment. However, the designed emulator is capable of supporting also other scenarios and enable different isolation of CPU or network resources.
Data traffic generation: Traffic generation is performed by using the Iperf network testing tool. Tests were performed by establishing a connection between an Iperf

server instance running on each APP and listening on the ogstun TUN interface of the MEC, and the Iperf client on the EU side, which generates traffic towards the APP. A helper class has been defined to ease the traffic generation, which allows to select a specific slice, the protocol to use (TCP or UDP), the port, uplink/downlink traffic generation, the bit-rate to generate (in Mbps) and its duration.

CPU load generation: To generate static or dynamic CPU load of an APP, we leverage the multiprocessing package to generate artificial CPU load by running one process for each online CPU core of the DockerHost hosting the APP. The amount of generated load is controlled by monitoring the container's CPU usage (using the psutil package) and regulating the rate of the artificial calculations to meet the CPU load requested. Since the psutil package returns the CPU load of the system, a re-scaling considering the resources allocated to the DockerHost is performed.

II.E. Monitoring

To monitor the scenario and collect results we implemented a Monitor class, which continuously oversees the data traffic and the CPU load monitoring.

After its instantiation, the Monitor class can be used to start/stop the monitoring of the throughput as well as the CPU of the desired hosts and applications. In the following, we explain more in detail how those monitors are implemented.

Data traffic monitoring: Throughput measures are collected for the overall traffic handled by a MEC as well as for each established Iperf session of the APPs. In the former case, a process is deployed in each MEC node (i.e. its container) and listens on a dedicated Redis channel. Upon the reception of the activation message, this process sends the overall throughput measured on the ogstun interface directly to the Redis database.

CPU Monitoring: Several tools are available for CPU monitoring (e.g.: Glances[5]). However, since most of them leverage psutil, they incur the issue to disregard the containers' CPU constraints that are emulated (as previously explained). For this reason, we preferred to follow a lower level approach and leverage the Docker Client APIs to retrieve the container CPU load through the Docker stats. To realise this, the class Monitor() implements a utility function which starts the CPU monitor for a specific container. Every time this function is invoked, a separate thread is started, which collects and tracks the container CPU status.

It is worth noticing that while measuring a MEC node we want to obtain its overall CPU usage. Conversely, measuring an APP CPU usage, the aim is to obtain its CPU usage over its hosting MEC node. For this reason, the measured CPU load of an APP is re-scaled with respect to the CPU resources allocated to its parent container.

[5] https://nicolargo.github.io/glances/

III. Experimental Results

The environment used for the emulation consists of a VM with 2 CPU allocated on a standard laptop with 8 x Intel Core i7-8665U CPU @ 1.90GHz with 16 GB RAM. To calibrate our emulated environment with real hardware we use the Passmark tool providing the CPU mark, a CPU performance measure. Figure 2(a) illustrates how to tune the MEC node resources to the desired value. For example, if we want to tune the MEC performance to be comparable with a RaspPi Model 3, then around 37% of one CPU should be allocated, while three RasPis can be achieved with roughly 90% of one CPU allocated.

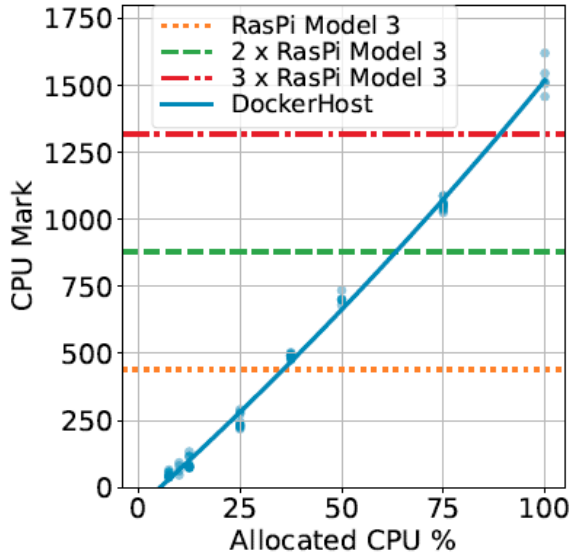

Fig. 2 (a) Performance comparison between DockerHost and RasPi Model 3.

Scenario 1: To test whether the system is working as expected and to understand the system's limitations, in this scenario, one UPF and one APP are deployed in MEC to emulate CPU load. Both applications exchange data traffic with one UE. Results are provided in Figure 2(b) where six zones are highlighted, corresponding to different operational setups of the system: (1) CP configuration, (2) UE deployment, (3) the APP is instructed to use 80% of the MEC CPU resources, (4) MEC container is forced to use a maximum of 37.5% of its allocated resources, (5) the UE generates from 40 to 70 Mbps of data traffic, with 1 full CPU allocated to the MEC node, (6) same as (5) but MEC resources are set to 37.5%. The corresponding throughput degradation starting from 50 Mbps of offered traffic clearly outlines the effect of the resources limitation.

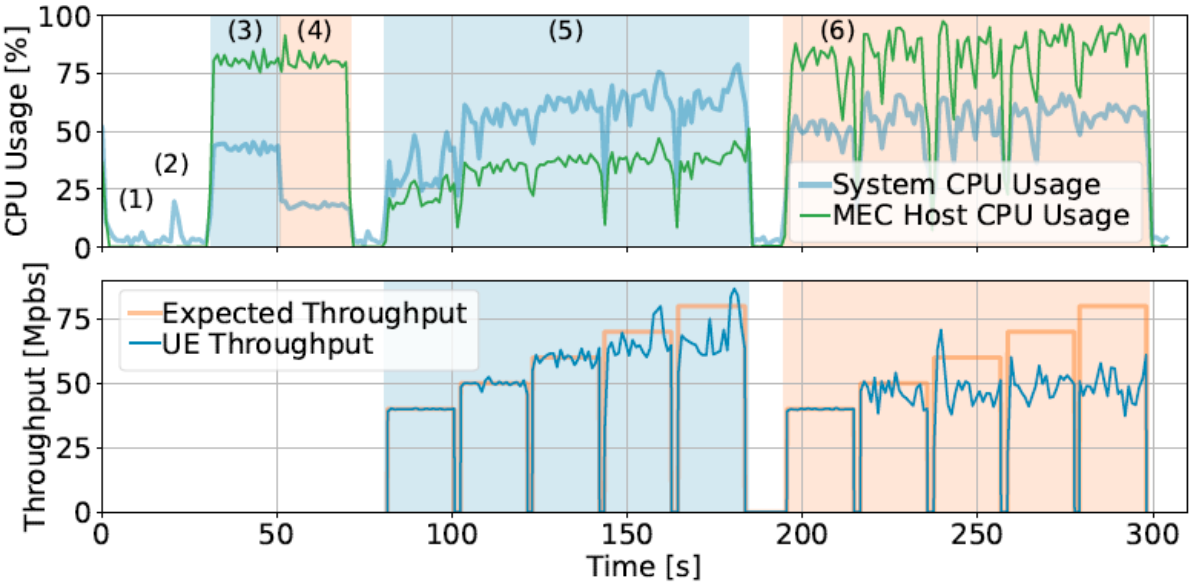

Fig. 2 (b) Performance evaluation in a dynamic operating scenario.

A more comprehensive performance evaluation under heavy traffic is shown in Figure 2(c) where we compare the throughput handled by the MEC with its CPU usage under different MEC resource constraints. As shown before, we can observe that when the MEC CPU resources are bounded to 100 and 70%, we reach the system limit as the throughput falls between 70 and 80 Mbps (the maximum level supported with a single CPU). Conversely, when the MEC CPU resources are bounded to 37.5 and 20% the throughput is limited by the scarcity of available MEC resources.

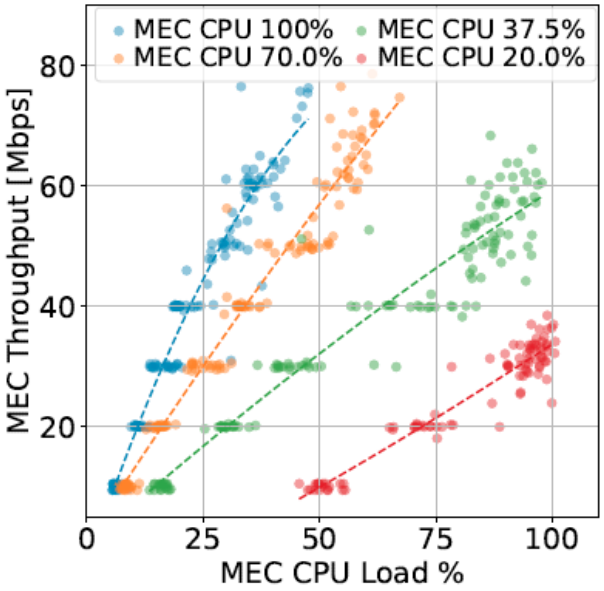

Fig. 2 (c) Throughput vs CPU load for variable resource availability.

Scenario 2: To study the impact of communication and computational intensive APPs, we deploy in MEC the UPF, an APP1 handling 40 Mbps of data traffic generated by

one UE, and APP2 using from 0 to 90% of the MEC computing resources. We can observe from Fig 3(a) that, when 100% of the CPU is allocated to the MEC, the level of CPU usage of APP2 does not impact on the achieved throughput. Conversely, with 20% of available CPU resources, we observe relevant a throughput decrease as soon as APP2 starts computing tasks.

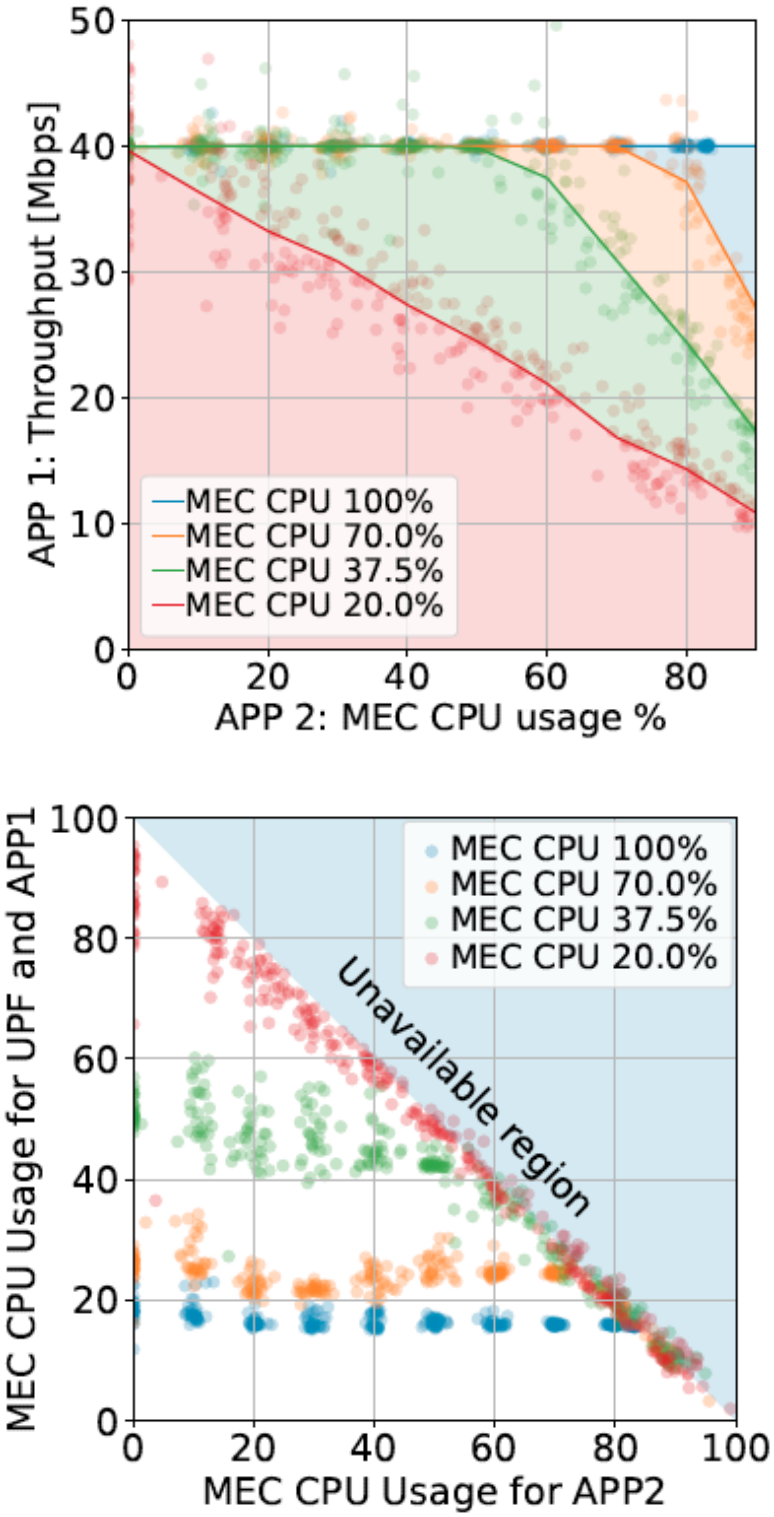

Fig. 3. Performance evaluation of a MEC hosting one throughput-intensive APP coexisting with a CPU-intensive APP.

This trade-off between CPU and data rate defines two different areas of the resulting diagrams (Fig. 3 (top) and Fig. 3 (bottom)): the area under the curve related to a specific scenario (and corresponding resource limitations) represents the possible operating points and tradeoffs achievable by the system, while the rest of the diagram cannot be achieved due to resource starvation.

IV. Conclusions

In this work, we proposed an environment to analyze MEC performance in an E2E 5G System.

The framework has been implemented to deploy the network and flexibly perform the experiments, using well-known open source software and sample dockerised applications to emulate a generic application handling data traffic and performing computing tasks. Results demonstrate that we can effectively emulate resource constrained MEC nodes and study APPs performance under high data rate and CPU usage.

Such characterization of the MEC performance tradeoffs will enable to (1) dimension the MEC node resources depending on the expected operating scenarios and related applications KPIs, and (2) properly allocate or migrate services and maintain the desidered KPIs. Both scenarios will be subject of further work on the topic.

Acknowledgements

This work was partially funded by Provincia di Trento and Oxys Solutions in the framework of the E2E fiber&5G project (L. 6/99).

References

[1] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P.-V. Mekikis, A. Antonopoulos, and C. Verikoukis, "Online VNF Lifecycle Management in an MEC-Enabled 5G IoT Architecture," IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4183–4194, 2020.

[2] D. Harutyunyan, R. Fedrizzi, N. Shahriar, R. Boutaba, and R. Riggio, "Orchestrating End-to-end Slices in 5G Networks," 15th International Conference on Network and Service Management, CNSM 2019, 2019.

[3] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "VNF placement optimization at the edge and cloud," Future Internet, vol. 11, no. 3, pp. 1–23, 2019.

[4] Q. Ye, W. Zhuang, X. Li, and J. Rao, "End-to-End Delay Modeling for Embedded VNF Chains in 5G Core Networks," IEEE Internet of Things Journal, vol. 6, no. 1, pp. 692–704, 2019.

[5] D. Boru Oljira, K. J. Grinnemo, J. Taheri, and A. Brunstrom, "A model for QoS-Aware VNF placement and provisioning," in 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFVSDN 2017, 2017, pp. 1–7.

[6] A. Garcia-Saavedra, G. Iosifidis, X. Costa-Perez, and D. J. Leith, "Joint optimization of edge computing architectures and radio access networks," IEEE Journal on Selected Areas in Communications, vol. 36, no. 11, pp. 2433–2443, 2018.

[7] G. Nardini, G. Stea, and A. Virdis, "Scalable real-time emulation of 5G networks with Simu5G," IEEE Access, vol. 9, pp. 148 504–148 520, 2021.

[8] C. Fiandrino, A. Blanco Pizarro, P. Jimalenez Mateo, C. Andrales Ramiro, N. Ludant, and J. Widmer, "openLEON: An end-to-end emulation platform from the edge data center to the mobile user," Computer Communications, vol. 148, no. September, pp. 17–26, 2019.

[9] Z. Xiang, S. Pandi, J. Cabrera, F. Granelli, P. Seeling, and F. H. Fitzek, "An Open Source Testbed for Virtualized Communication Networks," IEEE Communications Magazine, vol. 59, no. 2, pp. 77–83, 2021.