MECPerf: An Application-level Tool for Estimating the Network Performance in Edge Computing Environments

Chiara Caiazza[‡][*], Leonardo Bernardi, Marco Bevilacqua, Alessandro Cabras, Claudio Cicconetti[†], Valerio Luconi[†], Gabriele Sciurti, Elisabetta Senore, Emilio Vallati, Alessio Vecchio[*]
[*]Dip. di Ingegneria dell'Informazione, University of Pisa, Italy, chiara.caiazza@phd.unipi.it, alessio.vecchio@unipi.it
[†]IIT-CNR, Pisa, Italy, claudio.cicconetti@iit.cnr.it, valerio.luconi@iit.cnr.it
[‡]University of Florence, Italy, chiara.caiazza@unifi.it

***Abstract*—Edge computing is an emerging architecture in 5G networks where computing power is provided at the edge of the fixed network, to be as close as possible to the end users. Computation offloading, better communication latency, and reduction of traffic in the core network are just some of the possible benefits. However, the Quality of Experience (QoE) depends significantly on the network performance of the user device towards the edge server vs. cloud server, which is not known *a priori* and may generally change very fast, especially in heterogeneous, dense, and mobile deployments. Building on the emergence of standard interfaces for the installation and operation of third-party edge applications in a mobile network, such as the Multi-Access Edge Computing (MEC) under standardization at the European Telecommunications Standards Institute (ETSI), we propose MECPerf, a tool for user-driven network performance measurements. Bandwidth and latency on different network segments are measured and stored in a central repository, from where they can be analyzed, e.g., by application and service providers without access to the underlying network management services, for run-time resource optimization.**

***Index Terms*—Edge computing, Network measurements, 5G.**

I. Introduction

5G is progressing on its way to revolutionize the way people and objects will communicate. One of the main steps that have been achieved is about its architecture: the formerly monolithic core network, owned and operated by a single telco operator, has been transformed into a virtualized flexible infrastructure, possibly multi-tenant. Furthermore, *edge computing* has emerged as a paradigm to allow the execution of services and applications as close as possible to the users, which has clear benefits in terms of lower application latency and reduced network traffic, compared to a cloud deployment in data centers [1], [2]. These advantages are especially appealing to some vertical segments, which in fact have already established alliances to define reference architectures and harmonize efforts with actors across their respective value chains: industrial Internet [3], [4], connected vehicles [5], smart enterprise [6].

The European Telecommunications Standards Institute (ETSI) has recognized the need for standardization in this area and founded an Industry Study Group (ISG) on Multi-access Edge Computing (MEC), which attracted so far more than 100 participants [7]. The group has identified 35 use cases of high business relevance [8], based on which they have specified a reference architecture and open Application Programming Interfaces (APIs), which fully embrace the ongoing softwarization trend in 5G. A MEC domain example is illustrated in Fig. 1: edge networks are close to the User Terminals (UTs) and include the *MEC hosts*, which have a virtualized computation/storage infrastructure that can be operated by a common Virtual Infrastructure Manager (VIM) via the so-called `Mm7` interface. Third-party service providers can on-board their applications (*MEC apps*) by means of standardized interfaces that offer interoperability with all vendors of network equipment supporting ETSI MEC. The MEC apps are managed via the `Mm5` interface by a Mobile Edge Platform Manager (MEPM), which also operates the *MEC services*: these are platform-installed special applications that provide MEC apps with information on users (e.g., location) and infrastructure (e.g., radio network status), and also provide means to influence operation (e.g., create traffic steering rules). The MEC services are not illustrated in the figure because they are only marginally relevant to this work. Finally, the *MEC orchestrator* is the component in charge to take all decisions on the lifecycle of MEC apps: when to start/stop/migrate them or the mapping between UTs and MEC hosts.

Once an application in the UT has created a context and a MEC app has been provisioned in the MEC host that is deemed most suitable at the given time, there are potentially three data paths (see Fig. 1): (1) between the UT and the MEC app, traversing the access and networks (this is the shortest-latency path); (2) between the UT and the cloud, also passing through the core network and Internet; and, (3) between the MEC app and the cloud. Which path is used for what operation depends on the application and it is not under the control of the telco operator. On the other hand, the ETSI MEC does not provide a specific API or service to query the instantaneous network-layer performance of any of the three paths[1] However, for many applications that are either latency-

[1]Admittedly, the standard *does* allow an application to specify Service Level Agreement (SLA) guarantees upon context creation, but the way how there are handled and what happens in case they are (temporarily) violated is left to the equipment manufacturers.

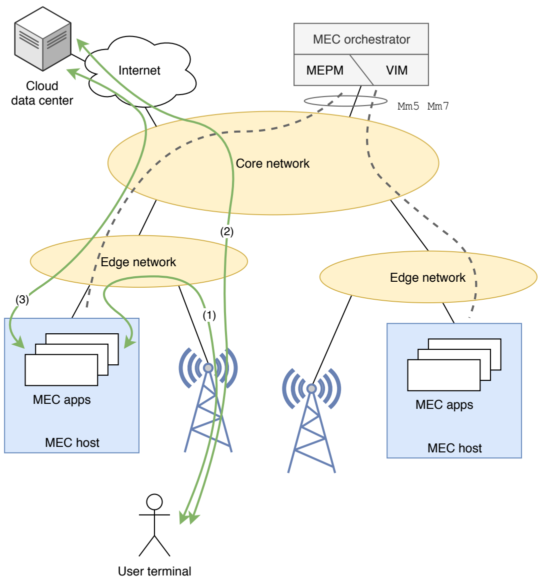


Fig. 1: Edge computing with ETSI MEC.

sensitive or bandwidth-intensive, it would be very useful to have such information at hand to take appropriate decisions. For this reason, in this work we propose a methodology to perform network-level measurements that is integrated with the ETSI MEC architecture.

In this paper we propose MECPerf, a system designed to measure network performance metrics in a MEC-enabled infrastructure. MECPerf is able to collect the network performance on the highlighted network segments in different ways: using active or passive techniques, or through self-measurements of KPIs by applications. Results are stored in a central repository so that they can be subsequently aggregated and analyzed. The final goal is to allow application and service providers to tune application/service execution parameters according to the instantaneous network conditions, which would not be available otherwise beyond the realm of the mobile network operator.

II. Related Work

Some works tried to quantitatively evaluate the benefits achieved when computing elements are shifted in proximity of the end users.

In [9], a fog computing architecture for the healthcare environment is compared to a traditional cloud computing architecture. By processing the data where they are produced, a reduction of traffic between layers can be achieved, together with reduced energy consumption and lower latency.

The benefits of edge computing, compared to cloud computing, were quantified by observing the response time and energy consumption of applications in multiple scenarios (processing-intensive and latency-sensitive) [10]. Results showed that edge computing can bring evident performance benefits, especially when the location of the edge node is close to the end user.

The SOUL framework shifts the computational load of sensor-based applications to the edge in a transparent way for programmers [11]. Micro-benchmarks have shown that SOUL improves overhead, scalability, and power consumption compared to the standard approaches.

In vehicle-to-pedestrian systems, cars and users' smartphones periodically generate contextual information that can be used by a collision detection algorithm to prevent dangerous situations [12]. The study shows costs and benefits in terms of battery consumption and processing time when migrating tasks from the smartphones to the edge nodes. Other studies about costs and benefits of migration of an application from the remote cloud to the edge nodes can be found in [13] and [14].

Latency-aware placement of service function chains has been studied in [15]. The problem has been faced using Integer Linear Programming (ILP) techniques, and the model assumes that information about the capacity of links connecting edge nodes and centralized cloud elements is available. This highlights the importance of network measurement tools like MECPerf, which are able to provide run-time information about network conditions. The availability of network bandwidth information on the relevant network segments is also required in [16], where a hierarchical edge approach is considered.

All the works mentioned assume that performance is determined exclusively by an appropriate orchestration of computation resources, generally ignoring the impact of network conditions. However, in heterogeneous and dense networks, runtime conditions could play an important role in the overall user's Quality of Experience (QoE), especially for applications with stringent delay or large bandwidth requirements. Therefore, in this work we propose to *complement* the outcome of the works in the literature by incorporating real-time measurements on network conditions through MECPerf, which is introduced below.

III. MECPerf

MECPerf aims at collecting the network- and application-level performance indices in a MEC system.

A. Architecture and Operation

The architecture of MECPerf is shown in Fig. 2. The system is composed of four components: the MECPerf Client (MC) running on UTs, the MECPerf Observer (MO) running on MEC application servers, the MECPerf Remote Server (MRS) running on remote servers, and the MECPerf Aggregator (MA). MECPerf can perform measurements according to three different operational modes: i) active measurements, ii) passive measurements through traffic analysis, iii) self-measured indices collection. In the active measurements mode, the MECPerf components measure general performance indices about the network infrastructure (bandwidth and latency). In the passive measurements mode, MECPerf receives the traffic traces of applications running on the MEC and extracts the performance indices obtained by each application. Finally, in the third mode, a third-party application can measure its own performance indices and send them to MECPerf via an open

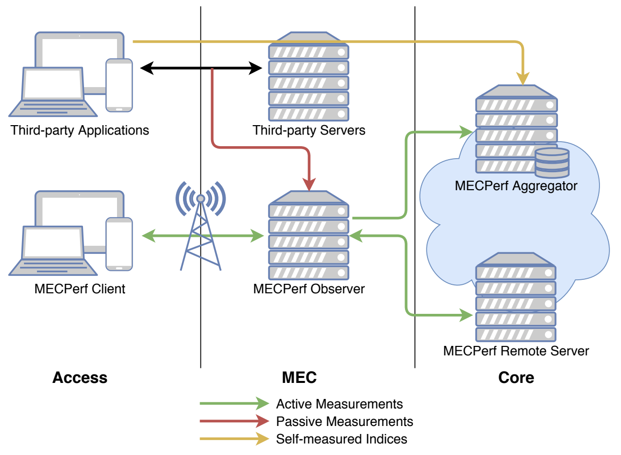


Fig. 2: MECPerf architecture.

interface. Not all components are involved in all the three operational modes.

1) Active Measurements: As introduced above, an application using computation offloading may interact at any given time with either a MEC server or a remote server in the cloud, in the so-called edge-cloud paradigm. Therefore, the most relevant network segments are: i) the segment between the UT and the MEC server; ii) the segment between the MEC server and the remote server. By collecting the performance indices of the first segment, it is possible to quantitatively evaluate the suitability of a given MEC server to offer computation offloading services to the given user, as compared to using a remote server, resulting from the performance indices on the second segment. In practice, these real-time data regarding network conditions should be fused with information on, e.g., computation load and server usage, available to the service provider. Each time a new measure begins, the MC interacts with the MO to compute wireless link performance metrics, then the MO interacts with the MRS to perform the same measurement on the second part of the path. Finally, the MO collects the results and sends them to the MA, which stores the collected values in a database. MC and MO are located in different parts of the network and are able to collect measurements according to their own network perspective, as previously mentioned. The MA is needed to merge the measurements collected by multiple points of view into a single set of network performance indices that characterize the entire path.

2) Passive Measurements: In the passive measurement mode, just the MO and MA are involved. This mode is intended for measuring performance indices related to the execution of third-party applications running on the MEC. By exploiting functionalities available in the ETSI MEC standard, the MO can receive a duplicate of the traffic generated by applications running on the MEC, from which it can extract the desired performance indices in real-time. It must be noted that the performance indices computed in this way are those observed from an external point of view, which is unaware of the application dynamics. The measured indices are periodically uploaded to the MA .

3) Self-measured Indices Collection: MECPerf offers third-party applications the possibility to collect and report their own metrics via an open interface provided by the MA. This is the most accurate way to obtain the performance indices about third-party applications, however not all application providers could be willing to implement their own measurement methods, for various reasons. Hence, the passive measurement mode is also needed.

B. Implementation

1) Active Measurements: MECPerf collects the following network metrics: TCP bandwidth, UDP bottleneck capacity, and latency, both TCP-based and UDP-based. TCP bandwidth is just the bandwidth of a data stream transfer, as observed by the receiver. The bottleneck capacity is defined as the capacity of the narrowest link over a certain network path [17]. In other words, the bottleneck capacity is an estimation of the maximum obtainable throughput over a path. To compute the bottleneck capacity MECPerf uses a known technique based on UDP packets that are sent back to back [17], [18]. The latency between two end-points is computed by sending a message and waiting for the response, then computing the round-trip time (RTT). Each metric can be taken for the two communication directions. Precisely, we distinguish between uplink measures, i.e. from the MC to the MO and from the MO to the MRS, and downlink measures, i.e. from the MRS to the MO and from the MO to the MC. The measurement functionalities are provided as a Java library, which is shared between the MC, the MO, and the MRS. The MA uses a MySQL database to store the collected metrics, while a Python Flask server makes them available to the application/service providers through a REST interface. Two prototype implementations of the MC are currently available. The first is a command-line application, useful for automated tests, while the second is an Android app, useful for on-site data collection guided by human operators. In a real system, the MC should be embedded in the mobile app running on the client device and under the control of the application provider.

2) Passive Measurements: The performance indices that can be collected via passive measurements are TCP bandwidth and latency, and UDP bandwidth. For each flow directed to one of the monitored applications, the MO stores a *(timestamp, payload size)* pair for each packet, which can be then used to compute the throughput of that flow. It must be noted that MECPerf ignores completely the payload content. Just for TCP flows, the MECPerf Observer is able to compute the experienced latency by using ACK numbers of the TCP protocol. Every time a valid ACK is seen, the MECPerf observer finds the ACKed packet, and computes the latency as the difference between the time of arrival of the ACK packet and the ACKed packet. Latency is stored as couples *(timestamp, RTT)* for each ACKed packet. In this case, the timestamp is related to the packet carrying the ACK. This feature is not available for UDP flows, as there is no way for

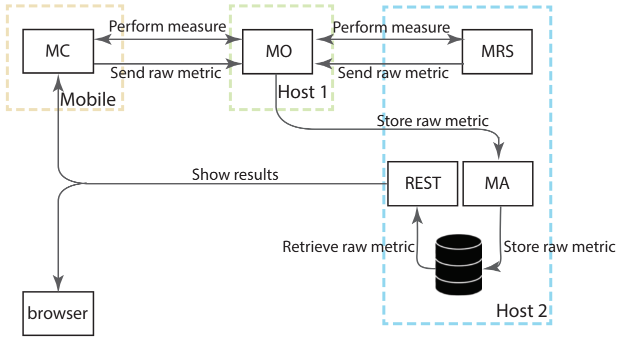


Fig. 3: Validation setup

an external point of view to correlate incoming and outgoing packets without knowing details of the application internal protocol, as UDP lacks mechanisms that can be used for such purpose (such as flow control).

3) Self-measured Indices Collection: The collected measurements are application-level bandwidth and latency. The application that is willing to share its own performance indices can choose which metrics to provide. Metrics are provided via the MA REST interface with HTTP POST requests. The measurements should be formatted as a JSON object containing relevant fields as IPs and ports of the end-points of the measured application, an alphanumeric identifier for the service (e.g., YouTube, Netflix), the protocol that can either be TCP or UDP, and finally two objects that contain uplink and downlink measurements. These objects contain a list of (key, value) pairs, where the key is the timestamp of the measured value, and the value is the measured metric, which can be either bandwidth or latency.

The code of MECPerf is open source and publicly available on GitHub[2].

C. Compatibility with NFV-based Architectures

During the second 5GCity hackathon [19], we implemented the MECPerf components as virtual machines, which were then wrapped as virtual network functions (VNFs) and used to build a network service (NS). The VNFs were built using the 5GCity SDK, a tool to develop VNFs compatible with cutting edge technologies for managing and orchestrating 5G/NFV infrastructures, such as Openstack Virtualized Infrastructure Manager (VIM) and Open Source Management and Orchestration (MANO), which is the ETSI-hosted project to develop an NFV Management and Orchestration software stack aligned with ETSI NFV.

IV. Experiments

In this section we provide some preliminary results of measurement campaigns that we ran using the MECPerf Active Measurements mode.

A. Validation

The implementation of the latency and bandwidth mechanisms was experimentally validated in a controlled setup. We installed the MECPerf components on two machines connected to the same network at the University of Pisa. More precisely, we deployed the MO on the first machine (Host 1) and the MA, the MRS, and the REST server on the second one (Host 2). Finally, we used a smartphone connected to the same university network via Wi-Fi to run the client application. The experimental setup is summarized in Fig. 3. We introduced artificial delay and bandwidth restrictions between the two hosts, and we then verified if the measured values were consistent with the set ones. To this purpose, we used *tc-netem*, which configures the traffic control policies on the kernel packet scheduler [20], [21]. More precisely, for latency tests, artificial delays were applied to the outgoing interface of Host 2. Instead, for bandwidth tests, rate limits were applied to the outgoing interface of Host 1 and Host 2 for uplink and downlink tests respectively. The results presented in the following are the average of ten repetitions.

Fig. 4 shows the latency measured between the MO and the MRS when using increasing artificial delay values, from 0 ms up to 200 ms. Substantial differences between the delay applied and the measured latency cannot be observed.

To validate the bandwidth estimation mechanisms, we applied an artificial rate limit r_{pkt} to the outgoing interfaces of Host 1 and Host 2, depending on the direction of the test to be carried out. Observed values cannot be directly compared with the applied bandwidth limitations, as the latter ones operate at the data link layer while the bandwidth in MECPerf is measured at the application layer. If a r_{pkt} limit is imposed at the data link layer, the bandwidth available at the application layer should be equal to

$$r_{app} = \frac{L_{app}}{L_{pkt}} \cdot r_{pkt} \tag{1}$$

where L_{pkt} is the total length of data link packets and L_{app} is the amount of application-layer data in each packet.

UDP bandwidth tests were configured to send datagrams with a payload of 1024 bytes ($L_{app} = 1024B$). As L_{app} is smaller than $MTU - H_{ip} - H_{udp}$, where H_{ip} and H_{udp} are the lengths of the IPv4 and the TCP headers, we can assume that UDP packets were not fragmented. Finally, the packet length of each UDP datagram can be computed as

$$L_{pkt} = L_{app} + H_{udp} + H_{ip} + H_{eth} + T_{eth} \tag{2}$$

where H_{udp} is the length of the UDP header, H_{eth} is the length of the Ethernet header, and T_{eth} is the length of the Ethernet trailer.

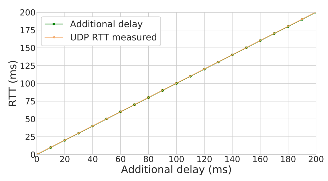
In TCP tests, a stream of bytes is sent in a short amount of time. As a consequence, we considered each TCP packet filled at its maximum capacity, which is equal to the maximum transmission unit (MTU) minus the space reserved for the headers. Thus, the L_{app} is supposed to be equal to

$$L_{app} = MTU - H_{ip} - H_{tcp} \tag{3}$$

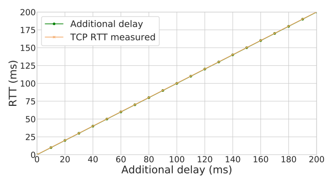
where H_{tcp} is the length of the TCP header. For TCP, we considered L_{pkt} equal to

$$L_{pkt} = MTU + H_{eth} + T_{eth} \tag{4}$$

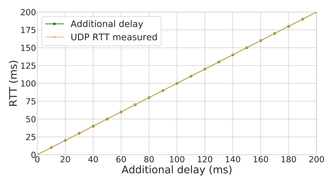
[2]https://github.com/MECPerf/MECPerf



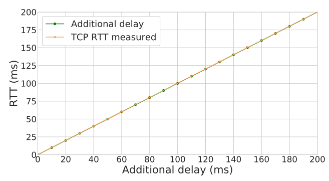
(a) Uplink UDP latency test



(b) Uplink TCP latency test

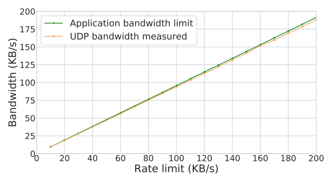


(c) Downlink UDP latency test

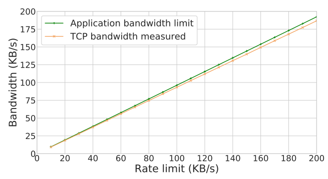


(d) Downlink TCP latency test

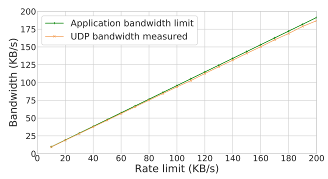
Fig. 4: Validation tests for UDP and TCP RTT estimation mechanisms.



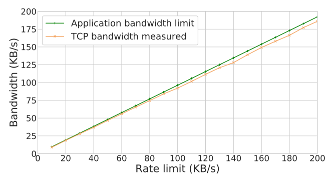
(a) Uplink UDP bandwidth test



(b) Uplink TCP bandwidth test



(c) Downlink UDP bandwidth test



(d) Downlink TCP bandwidth test

Fig. 5: Validation tests for UDP and TCP bandwidth estimation mechanisms.

The following values were considered: 1500 bytes for the MTU, 20 bytes for IP and TCP headers, 8 bytes for the UDP header, 14 bytes for the Ethernet header, and 4 bytes for the Ethernet trailer. In the end, $r_{app} = 0.957 \cdot r_{pkt}$ for UDP and $r_{app} = 0.962 \cdot r_{pkt}$ for TCP.

Bandwidth limitations were introduced with values of r_{pkt} in the 10-200 KB/s range. Fig. 5 compares the observed bandwidth with the expected r_{app} value. When high r_{app} values are considered, it can be noticed that TCP rates are slightly lower than the ones obtained in the corresponding UDP tests. This can be explained as the effect of the TCP slow-start mechanism, which slightly reduces the average throughput. Differences, between the measured bandwidth and the applied r_{app} limits, are below 3%.

B. *Evaluation*

We run a small-scale experimental evaluation of MECPerf on a real network. The key network performance indices were collected on both a wireless segment and a wired one using MECPerf. Collection of performance indices was carried out every 30 s for approximately four hours. The MC was executed on a normal PC, connected via Wi-Fi to a LAN of the University of Pisa, Italy. On the same LAN, the MO was executed on a standard PC operating as a possible edge server. The MRS was executed on an Amazon AWS EC2 instance [22]. The MRS was physically located in the USA, east coast. TCP bandwidth was measured using 1 MB payloads, thus the obtained results can be representative of applications/services characterized by a communication scheme where the amount of transferred data is, on average, of such size.

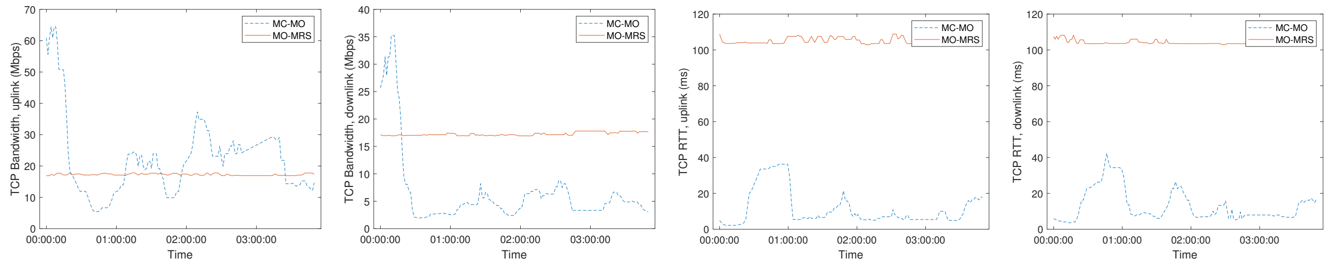
Fig. 6 shows the observed TCP bandwidth and RTT, on the two segments in both directions. On the left-hand side of Fig. 6a and 6b, the bandwidth on the MC-MO segment is greater than the bandwidth on the MO-MRS segment. This part corresponds to a period where the number of people in the building was rather small, and the load on the Wi-Fi access point was light. After approximately 25 minutes, the number of people in the building started to increase, because of a meeting involving approximately 60 persons. The increased load on the Wi-Fi infrastructure makes the observed downlink bandwidth on the MC-MO segment smaller than the bandwidth on the MO-MRS segment (Fig. 6b). In particular, the latter remains almost constant as the amount of traffic produced by the increased amount of people does not saturate the wired 10 Gbps links of the LAN. In the uplink direction this phenomenon is less evident (Fig. 6a), even though the difference between the first 25 minutes and the remaining period is still visible.

In practice, during the first period the bottleneck on the whole path is somewhere in the wired part, whereas during the second period the bottleneck is represented by the wireless access. Let us suppose that a bandwidth intensive application can be relocated from the remote cloud to the edge. During the first period, relocating the application in proximity of the users may bring some benefits. On the contrary, during the second period migrating the application is not going to provide any advantages, as the bottleneck is on the wireless access.

Fig. 6c and 6d show the situation in terms of TCP RTT. Also in this case, the impact caused by the increased number of people is still visible. However, for a latency-sensitive application, migrating close to the users may still bring some benefits (the latency on the MC-MO segment is always much better than the latency on the other segment). In this specific case, this may also be due to the fact that the remote cloud is reached via a trans-continental connection, and thus characterized by a significant latency.

V. Discussion and Conclusion

We see three possible business scenarios where the proposed methodology, an hence MECPerf, can be exploited by vertical players. First, a *vertical-telco cooperation scenario*, where



(a) TCP bandwidth, uplink (b) TCP bandwidth, downlink (c) TCP RTT, uplink (d) TCP RTT, downlink

Fig. 6: Experimental results.

the vertical installs MEC apps on MEC hosts which embed application-specific measurement modules. These modules send real-time streams of data to a collecting module in the core network, which is queried by the MEC orchestrator to take decisions and adapt the deployment of MEC apps so as to best fit the application-dependent KPIs, as opposed to relying merely on platform-related metrics. In this scenario the telco operator benefits from accurate measurements from the applications, and the vertical is provided with a better operation, and in the end higher QoE for its users. Second, if the vertical uses an *edge-cloud* application, i.e., one that implements computation offloading from the UT to either the edge or the cloud, then having real-time measurements from both the MEC hosts and the remote data center allows to tune more accurately migration or load balancing between the two. Third, we foresee that a third party may offer *measurements as a service*, by on-boarding MEC apps with the only purpose of gathering network-level measurements between MEC hosts and UTs, likely using active probing. This way, vertical service providers may take proactive decisions on whether a given context can or cannot be created for a UT at a given time, independent from guarantees that could be provided by the telco operator.

Acknowledgment

This work is partially funded by the Horizon 2020 Research and Innovation Programme under grant agreement No. 732638 (Fed4Fire+) through the 6th open call, and by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence). The views expressed are solely those of the authors.

References

[1] M. Campbell, "Smart Edge : The the Center of Data Gravity Out of the Cloud," *Computer*, vol. 52, no. December, pp. 99–102, 2019.

[2] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[3] Industrial Internet Consortium, "The Edge Computing Advantage," Tech. Rep., 2019.

[4] 5G-ACIA, "5G Non-Public Networks for Industrial Scenarios," Tech. Rep. March, 2019. [Online]. Available: www.5g-acia.orgwww.zvei.org

[5] 5GAA, "Toward fully connected vehicles : Edge computing White Paper," Tech. Rep., 2017.

[6] A. Reznik, A. Sulisti, A. Artemenko, Y. Fang, D. Frydman, F. Giust, H. Lv, S. Sheikh, Y. Yu, and Z. Zheng, "Mec in an enterprise setting: A solution outline," *ETSI, Sophia Antipolis, France, White Paper*, vol. 2, no. 30, pp. 1–20, 2018.

[7] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, K.-W. Wen, K. Kim, R. Arora, A. Odgers, L. M. Contreras, and S. Scarpina, "ETSI White Paper: MEC in 5G networks," ETSI, Tech. Rep. 28, 2018. [Online]. Available: www.etsi.org

[8] ETSI, "Multi-access Edge Computing (MEC); Phase 2: Use Cases and Requirements; ETSI GS MEC 002 V2.1.1," Tech. Rep., 2018.

[9] P. H. Vilela, J. J. Rodrigues, P. Solic, K. Saleem, and V. Furtado, "Performance evaluation of a Fog-assisted IoT solution for e-Health applications," *Future Gener. Comput. Syst.*, vol. 97, pp. 379–386, 2019.

[10] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, "Quantifying the Impact of Edge Computing on Mobile Applications," in *Proc. ACM SIGOPS APSys '16*, 2016, pp. 5:1–5:8.

[11] M. Jang, H. Lee, K. Schwan, and K. Bhardwaj, "SOUL: An Edge-Cloud System for Mobile Applications in a Sensor-Rich World," in *Proc. IEEE/ACM SEC '16*, 2016, pp. 155–167.

[12] Q.-H. Nguyen, M. Morold, K. David, and F. Dressler, "Car-to-pedestrian communication with mec-support for adaptive safety of vulnerable road users," *Comput. Commun.*, vol. 150, pp. 83 – 93, 2020.

[13] A. Napolitano, G. Cecchetti, F. Giannone, A. Ruscelli, F. Civerchia, K. Kondepu, L. Valcarenghi, and P. Castoldi, "Implementation of a MEC-based vulnerable road user warning system," in *Proc. AEIT AUTOMOTIVE '19*, 2019.

[14] M. Emara, M. Filippou, and D. Sabella, "MEC-Assisted End-to-End Latency Evaluations for C-V2X Communications," in *Proc. EuCNC '18*, 2018, pp. 157–161.

[15] D. Harutyunyan, N. Shahriar, R. Boutaba, and R. Riggio, "Latency-aware service function chain placement in 5g mobile networks," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, June 2019, pp. 133–141.

[16] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. IEEE INFOCOM '16*, 2016, pp. 1–9.

[17] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE Network*, vol. 17, no. 6, pp. 27–35, 2003.

[18] E. Gregori, A. Improta, L. Lenzini, V. Luconi, N. Redini, and A. Vecchio, "Smartphone-based crowdsourcing for estimating the bottleneck capacity in wireless networks," *Journal of Network and Computer Applications*, vol. 64, pp. 62 – 75, 2016.

[19] "Hack the 5G city: A new era of city services on 5G," https://www.5gcity.eu/hack_lucca, 21-22 November 2019.

[20] "tc (Traffic Control)," http://man7.org/linux/man-pages/man8/tc.8.html, Accessed on: March 2020.

[21] "NetEm - Network Emulator," http://man7.org/linux/man-pages/man8/tc-netem.8.html, Accessed on: March 2020.

[22] "Amazon Elastic Compute Cloud," https://aws.amazon.com/it/ec2/, Accessed on: March 2020.