Testing the resilience of MEC-based IoT applications against resource exhaustion attacks

Roberto Pietrantuono, Massimo Ficco, and Francesco Palmieri

Abstract—Multi-access Edge Computing (MEC) is an emerging computing model that provides the necessary on-demand resources and services to the edge of network, ensuring powerful computing, storage capacity, mobility, location, and context awareness support to emerging Internet of Things (IoT) applications. Nonetheless, its complex hierarchical model introduces new architectural interdependencies, which can influence the resilience of IoT applications against cyber attacks. Although application resilience has been investigated in the context of cloud computing, existing studies are not directly applicable to such an extended edge-cloud paradigm. The use of different enabling technologies at the edge of the network, such as various wireless access technologies and virtualization, implies several threats and challenges that make the analysis and deployment of resilience mechanisms a technically challenging problem. In this paper, we first present an overview of the threat model, describing the threats for the different layers of this paradigm. We then study the impact of resource exhausting attacks – a particularly relevant class for this paradigm - on three different IoT applications exploiting the services offered by the MEC-based architecture. We adopt a testing-based methodology conceived to characterise the resilience of such applications under attack. A set of most important resilience-related indicators are also identified. The characterisation's results are useful to support the analyst in planning proper protection means at individual architectural layers.

Index Terms—Internet of things, multi-access edge computing, resilience, threat model, resource-exhausting attacks

1 Introduction

To satisfy the growing demand for low latency, location awareness, and mobility typical of the IoT applications, new edge computing models have emerged. Multi-access Edge Computing (MEC) provides an IT service environment that extends the cloud computing capabilities to the edge of the mobile network, i.e, within the Radio Access Network (RAN) [1]. It offers service and cloud resources, such as computation, network, and storage, closer to the source of data, in order to fulfil more stringent end-to-end latency-sensitive, high-bandwidth and -computing demanding requirements of user applications. This results in offering context-aware applications and services, including IoT, augmented reality, intelligent video acceleration, mobile gaming, mobile crowd sensing, connected cars, etc.. According to Edge Computing Market report published by Meticulous Research in sept. 2020, the MEC market is expected to grow at a rate of 30.

Nonetheless, although this edge computing paradigm offers new opportunities and business models for IoT applications, MEC is not a mere extension of the cloud. Several layers of the implementation stack are logical and/or physical revisited, introducing additional security and resilience implications [2]. In the context of this work, resilience is defined as the ability of a system to provide an acceptable level of service in the presence of external perturbations such as security attacks [3]. Although application resilience to security attacks has been widely discussed in the context of the cloud, it has not been likewise investigated in the MEC-IoT context. In MEC-IoT, difficulties arise because of the multiple interactions with different access technologies, including Long Term Evolution (LTE)/5G, WiFi, Bluetooth, etc. This extensive use of wireless technologies on the edge of networks makes the infrastructure more prone to several types of attacks, such as Denial-of-Service (DoS) and wireless jamming exploited to consume the communication and computing resources at the edge. Man-in-the-middle attack can be used to inject or eavesdrop traffic from the edge. Further problems arise during the provision of virtual resources, as cloud virtualisation must be extended beyond the cloud to reach the edge nodes [4], requiring the distribution of virtual machine (VM) images over public links. Malicious users may use these links to orchestrate a geographically coordinated DoS attack by a single or multiple compromised VMs, introducing communication delays aimed to make the edge nodes unavailable [5]. Finally, the complex MEC reference architecture, the different applications, and the multi-tenant infrastructures make the analysis and deployment of resilience mechanisms across the architectural layers a technically challenging problem. Therefore, resilience represents a relevant hard-to-satisfy requirements during the MEC-based IoT application provisioning.

In this paper, we first present an overview of the threat model that may affect the different layers of an extended-cloud architecture based on an MEC model, so as to highlight the security and resilience challenges. Then, since this computing model is more vulnerable to threats resulting from malicious load fluctuations, and resource exhausting attacks, we investigate the impact of such attacks on three different MEC-based IoT applications. To this aim, we define

- *M. Ficco and F. Palmieri are with Department of Computer Science of the University of Salerno, I-84084, Fisciano (SA) - Italy. E-mail: mficco@unisa.it, fpalmieri@unisa.it.*
- *R. Pietrantuono is with Università degli Studi di Napoli Federico II, via Claudio 21, I-80125, Naples, Italy E-mail: roberto.pietrantuono@unina.it*

Manuscript received April 19, 2019; revised August 26, 2019.

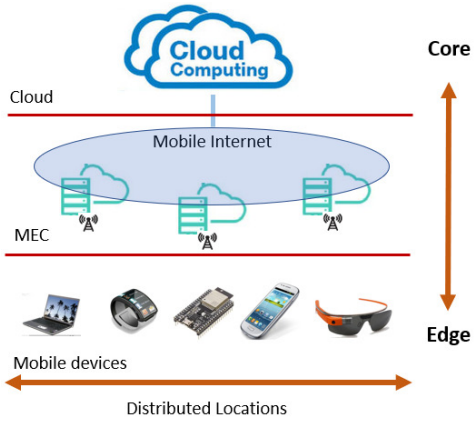


Fig. 1: Edge-IoT architectural model.

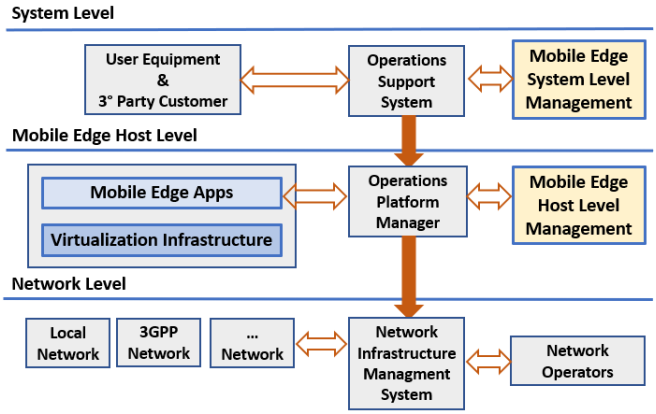


Fig. 2: The MEC reference architecture view.

a testing-based methodology to study the resilience from different perspectives. Through testing, measurements, and data analysis, the methodology allows: i) assessing the impact of attacks on the application resilience at different architectural layers, ii) investigating the propagation of the attack effects across layers, and iii) identifying resource-exhaustion indicators more related to resilience. The output can be used to support the analyst in planning proper protection/tolerance means at individual layers, as well as in developing attack detection strategies based on the identified resilience-related indicators. Since MEC potentially proves to be the groundbreaking technology in the field of IoT, the remote patient monitoring, the smart building control, and the remote surveillance have been considered as use cases to test the proposed methodology.

The rest of the paper is organized as follows: Sec. 2 presents an overview of the MEC paradigm, as well as the security and resilience related challenges. The objectives and proposed methodology to characterise the behaviour of the MEC-based IoT applications when subject to resource-exhaustion attacks are described in Sec. 3. The case studies and the adopted strategy are shown in Sec. 4. Experimental results are discussed in Sec. 5. Sec. 6 presents related work. Conclusions and future work are summarized in Sec. 7.

2 Edge-IoT Model

The considered edge-IoT model is shown in Fig. 1, in which the Multi-access Edge Computing is the layer that resides between the cloud and the mobile devices. Mobile devices and smart sensors are connected to the core network (i.e., the mobile Internet) through the edge network (i.e., the MEC). The MEC layer consists of geo-distributed virtual servers with built-in services specific for the target mobile environment. The servers are deployed at mobile user premises, for example, at a fixed indoor or outdoor location (e.g., a park, a shopping center, a smart building, a bus terminal) and on devices located in moving object (e.g., cars and buses). MEC can use wireless network elements for the communication, including the cellular base stations, WiFi access points, and low power cellular base stations (i.e., femto access points) [6].

2.1 Multi-access Edge Computing

MEC formerly known as Mobile Edge Computing, aims at providing the cloud computing capabilities and an IT service environment at the edge of the mobile network to reduce latency and offer higher bandwidth needed to improve the network operation efficiency and service delivery [7]. The MEC environment consists of MEC virtual nodes deployed at the edge of the mobile network needed to support of LTE/5G base stations and multi-Radio Access Technology (3G/LTE/WLAN) cell aggregation sites located within outdoor and indoor environments [8]. The functional elements described in the MEC framework provide support to the offered services, such as application execution, radio network information, and location awareness. The framework describes the main functional blocks of the MEC environment and the reference architecture, as well as defines the reference points between them. As Fig. 2 shows, the involved entities are grouped in three levels: host level, network level, and system level. The management service is splitted across the different levels of the MEC architecture. Specifically, the host level includes the Mobile Edge (ME) host and ME host management entities. The host level represents the middle level of MEC reference architecture. The ME host is composed of the virtualisation infrastructure, the ME platform, and the ME apps. The virtualisation infrastructure provides the computing, storage, and networking resources for running ME applications. The ME platform provides ME applications as services, and offers facilitates for running the ME applications on a particular virtualized infrastructure. It provides facilities to discover, advertise, and use the ME applications through a service registry, as well as DNS services and traffic rules implementation. The ME host management consists of a platform manager and a virtual infrastructure manager. At system level, the ME system level management acts as orchestrator, which is responsible of orchestrating the ME virtual infrastructure needed for application deploying. Moreover, it is responsible of relocation, termination and instantiation of applications, as well as authenticity checks and policy enforcement for the ME application. Finally, the network level provides the connectivity between external and internal MEC entities.

2.2 Security and resilience related challenges

As aforementioned, security represents one of the greatest challenges for the edge paradigm ecosystem [9], [10]. The different enabling technologies present in the edge paradigms, such as distributed and peer-to-peer systems, virtualization platforms, wireless networks, and wireless technologies require the orchestration of diverse security mechanisms to protect all these building blocks. On the other hand, assuring the security of each single enabling technologies does not assure the security of the whole system. Move cloud capabilities to the edge of the network (e.g., migrating services, heterogeneous edge data-centres collaboration) involves new security threats, whose have not been widely studied [11]. Moreover, such security threats that affect specific features of edge paradigms and their building blocks could be inherit by the whole system. A clear example of this is the Internet of Things, which represents one of the major use case in edge paradigm. IoT combines multiple layers of technologies (from network to cloud to mobile) and provides connectivity in a heterogeneous ecosystem, generating a considerable attack surface that can affect MEC-based systems [12].

Therefore, although resilience has always been a great challenge for cloud computing, it is even more challenging for the MEC-based IoT applications, due to interactions with various wireless access technologies, multi-tenant infrastructures, virtualization, and different MEC applications, which make technologically difficult the implementation and deployment of resilience mechanisms [2], [13].

2.2.1 Threat model

In order to define the specific threat model that affects MEC-based IoT paradigm, it is necessary to analyze how the lack of a global perimeter affects the security of edge ecosystem. The presence of a single mobile network operator, which controls part of the core network and several geographical distributed edge data-centres (i.e., the mobile network infrastructure) allows enforcing consistent security policies that limit the chances of an intruder to compromise or control part of the service infrastructure. On the other hand, such an extensive ecosystem will not be controlled by one single owner. Moreover, small companies like stores can also deploy their own micro edge data-centres, and allow their users to become active participants in the provisioning of services, which makes this ecosystem more heterogeneous and vulnerable. Therefore, the lack of a well-defined perimeter makes this paradigm the target of external attackers (which could control several infrastructure elements), and malicious internal users that cloud control elements of the infrastructure (such as user devices, servers, virtual machines, edge data-centers, and sections of the network) with which they could influence other sections of the infrastructure or provided services. For example, the "anywhere" principle of IoT causes the geographical distribution of the edge entities (e.g., MEC servers), so as to provide services in proximity to local source of data (e.g., mobile devices, entities inside a building). A consequence of this extended scope is that each component of the infrastructure can be indirectly targeted by malicious users at any moment. Deploying malware in a virtual machine, hackers could try to gain more privileges for exploiting further vulnerabilities, and take control of core network elements, in order to compromise large sections of the whole ecosystem. If an attacker compromises an edge data-centre, it could control the services provided in neighbouring geographical locations.

Moreover, MEC paradigms can use microservers (e.g., Raspberry Pi, user devices) to provide their services. On these devices, the security policies might not be properly implemented or maintained [15]. For example, MEC paradigm allows the creation of clusters of devices at the very edge of the network for providing services (e.g., for exploiting parallel mechanisms), which can be used to inject malware and perform DoS attacks against honest participants.

Finally, although most threats affecting edge paradigms are the same that target traditional data-centres, server farms and networking infrastructure, the specific decentralized and distributed nature of MEC paradigm, and the provided services (such as the mobility support and the location awareness service), involve an impact different of these common threats, as well as novel threats and attacks will arise.

Roman et al. [15] present a threat overview across the different levels in MEC architecture. In particular, at infrastructure level, the edge network is part of the last-mile network. Therefore, different technologies are exploited to build the network, which make the MEC infrastructure prone to different attacks. For example, Denial-of-Service (DoS) and wireless jamming are common attacks used to consume the bandwidth and computing resources at the edge. Man-in-the-middle attack could be used to take control of a section of the network, and inject or eavesdrop traffic from the edge. It could be also used to gain access to the gateway network interface that interconnected 3G and wireless networks. Similarly, malicious gateway can be added to a cluster of nearby user devices to eavesdrop and/or inject traffic. Moreover, virtualisation adopted to share the resources among MEC applications could be exploited to deploy malicious virtual machines/malware for depleting the computing/networking resources shared between VMs. Finally, although the MEC core infrastructure is managed by the same mobile network operator that deploy the edge data-centres, the interactions with a cloud provider could be untrusted and represent the target of cloud-oriented attacks.

In this paper particular attention has been posed on resource exhausting attacks, which are tailored to hurt a specific point in the MEC-based IoT infrastructure, in order to inflict fraudulent resources consumption (e.g., in terms of CPU, memory, bandwidth, and energy), and analyze their effects on the architectural layers.

3 METHODOLOGY

3.1 Objective

The objective of the proposed methodology is to characterize the behavior of the MEC-based IoT applications when subject to cyber-attacks by using *testing*, *measurements*, and *data analysis*. In this case, the metric of interest is resilience. Resilience has been defined in several ways in the literature, depending on the context (e.g., [17] [18] [19] [20] [21]), but the underlying common concept is the same: it

represents *the ability to provide and maintain an acceptable level of service in presence of failures* [21]. The definition of resilience given by Smith *et al.*, although refers to networks, includes malicious attacks as possible faults against which an acceptable level of service should be maintained [18]. We are interested in specific kinds of failures, which are those induced by resource exhaustion attacks. Therefore, a MEC-based IoT application is not resilient with respect to resource exhaustion attacks if the attacks actually deplete the resources, and such resource depletion eventually impacts the user-perceived performance (hence compromising the mentioned "acceptable level of service"). Conversely, it is resilient if the resources are not impacted by the attack or they are impacted but their depletion does not affect the user performance (e.g, there are defense/tolerance mechanisms ensuring that the resource is not depleted beyond a certain limit). To assess resilience in a useful way for the analyst to take preventive actions (e.g., design defense/tolerance mechanisms), the system should be looked at as a grey- rather than a black box: a resource exhaustion attack can target nodes/devices at the three different layers in a different way and can affect various resources, depending on the type and target of the attack. Thus, the different degree of resilience to attacks within the architecture is the first useful indication we aim to provide. Moreover, the layers are clearly not independent from each other; it is expected that the degraded performance of an affected entity impacts the interacting entities in the other layers, eventually affecting also the user interface. Therefore, a secondary goal is to provide finer-grain indications about the possible propagation of the effects of an attack across layers. Both these outputs can give information for planning tailored protection/tolerance solutions. Finally, besides assessing resilience and the related propagation effects, detecting a resilience-impacting attack on time is a desirable goal too. The proposed methodology is a testing-time strategy, which does not aim to implement an attack detection mechanism, but we can still use the results of testing and measurements to give indications about which metrics can be more useful to predict (i.e., is more correlated to) resource exhaustion attacks, as well as can be used for attack detection strategies implementation. In summary, the methodology targets the following three sub-goals (SG):

- *SG1: Impact analysis*. The goal is to assess to what extent resource exhaustion attacks impact resilience and to understand which part of the cloud-extended architecture is more sensible for which attacks.
- *SG2: Propagation analysis*. The goal is to assess to what extent resource exhaustion caused by the attacks propagates to the user interface, namely to what extent they are perceived by end users.
- *SG3: Correlation analysis*. The goal is to understand what are the most useful indicators for detecting a resource exhaustion attack.

3.2 Overview

The basic idea to run the above analyses is to test the system against different resource exhaustion attacks, collect resource-related data during the attack, and analyze them offline according to SG1-SG3. To this aim, the following steps are carried out; the detailed activities of each step are described through case studies in the next Section.

1) *Security testing scenarios definition*. The first step is to define what attacks are used, referred to as *testing scenarios* (Sec. 4.2).
2) *Metrics definition*. This step defines what metrics should be gathered while running the testing scenario. In a MEC-based IoT application, each architectural layer (e.g., MEC server/gateway) is composed of more partitions (e.g., the OS subsystems of a gateway node, such as I/O or Network subsystem): given a partition p at the architectural layer l, we call it a *subsystem*, denoted as $S_{p,l}$. The goal of this step is to have metrics covering all the layers and partitions (Sec. 4.3).
3) *Tests execution*: Each testing scenario consists of: (i) a *pre-attack* phase, during which the system runs a pre-defined non-intensive workload (e.g., a workload foreseen by functional test scenarios); (ii) an *attack* phase, in which one or more specific attacks are launched; and (iii) a *post-attack* phase, in which the attack is stopped and the system is expected to perform like in the pre-attack stage (Sec. 4.4).
4) *Data collection*. During testing, data about the defined metrics are collected. At the end of testing, a set of time series (one per metric) of the same length is obtained (Sec. 4.4).
5) *Data analysis and interpretation*. Collected data are properly preprocessed and analyzed with respect to the three defined sub-goals (Sec. 4.5). This step aims at (i) synthesizing the behavior by computing the resilience for each subsystem and for the system as a whole (SG1); (ii) analyzing the propagation of the effect of the attacks to the end-user interface by looking at cause-effect relationships (SG2), and (iii) analyzing the (cor)relation between metrics and attacks in order to figure out what are the most useful indicators to detect a resource exhaustion attack (SG3).

4 Method walk-through via case studies

4.1 Experimental scenarios

The generalized MEC application can be modelled according to a services-based paradigm, with services deployed as separate entities, which interact via network calls to enforce separation and avoid tight coupling. The services are deployed at different architectural levels: client-side, edge level, and cloud. Each service exposes an application programming interface for other services to communicate.

Specifically, three experimental IoT applications are considered, including remote patient monitoring, smart building control, and remote surveillance by drones. In these applications, transferring a large amount of data to the centralized cloud servers may result in latency times that are not appropriate for the kind of IoT application. Therefore, MEC can offer specialized local services for processing capabilities for the large IoT data produced by the monitored contexts, extending functionalities to the edge of network and reducing the communication latency.

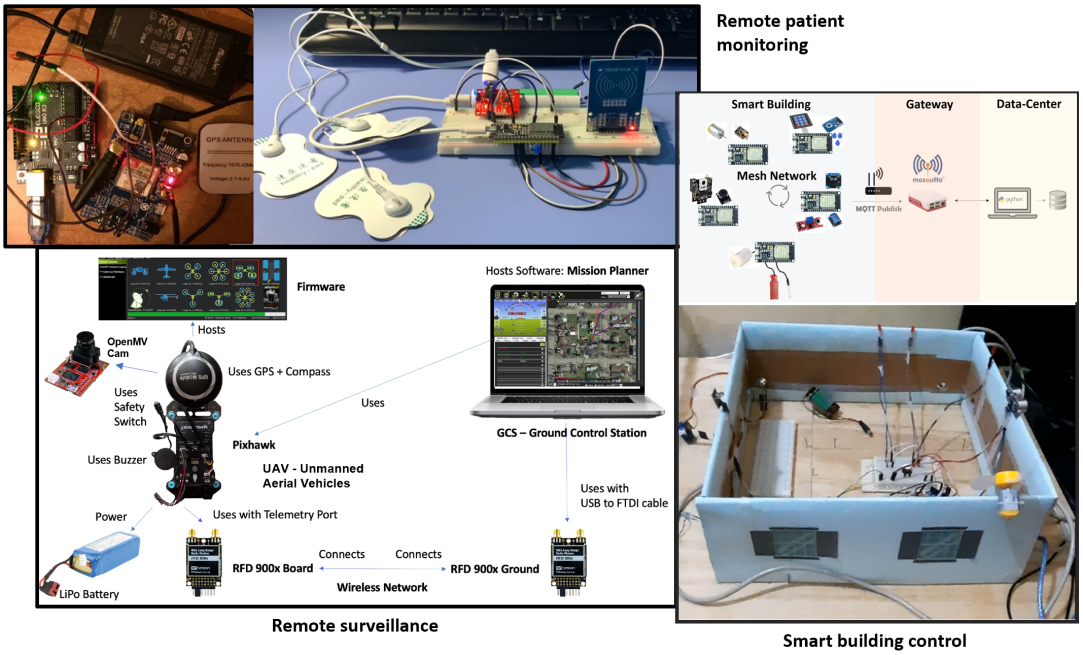


Fig. 3: The emulated MEC-IoT scenarios.

The remote patient monitoring consists in the automatic measurement of patient health signals, such as heart rate, blood pressure, temperature, etc. According to such a scenario, the client service collects data (i.e., vital signs) by the smart sensors worn by the user and forwards them to the service residing at the edge. It is also responsible for receiving the computed data from the edge services and to alert the patient. The service running at the edge is responsible for processing vital signs of the patients with severe illness or the elderly for real-time notification for medical practitioners, as well as providing specialized MEC services. For example, it may subscribe to the MEC Radio Network Information Service to react to the channel status of the patients or infer its position. Finally, the cloud service is responsible for collecting the statistics of patients.

The second experimental scenario consists of a smart building control system exploited to offer advanced services and produce significant savings through its ability to react quickly even to small changes that occur in the building (such as changes in air temperature, brightness, levels of occupation of the environments, opening windows, etc.). Compared to the previous scenario, the IoT infrastructure consists of a large number of different smart sensors distrbuted in the building. They are used for controlling and monitoring the different environments in which are placed, including gas presence, temperature, humidity, occupancy levels, security, home entertainment systems, opening and closing of doors and windows, etc.. The system is used to offer specialized local services for processing and storage the large IoT data produced within the building, expanding such functionalities to the edge of the network. Smart sensors connected to the IoT network become capable of sharing information with the edge and other sensors/actuators, and intelligently make automated alert and local adjustments at the optimal time. For example, if a high level of humidity is detected in the environment, an edge service could perform actions to increase air circulation (e.g., activates the ventilation fans or dehumidifiers, opens the windows, etc.).

Finally, the remote surveillance performed by means of Unmanned Aerial Vehicles (UAV), commonly called drones, is a latency-sensitive application. Specifically, it is implemented by a system composed of a drone, a ground station, and an edge server. The drone is in charge of analyzing the images acquired by a camera, as well as communicating with the edge server via the ground station (through a mobile link). Since the drone has limited computing resources and battery lifetime, reducing the computation workload during the flight is essential to save battery. Therefore, the drone can offload all or part of their computing tasks to remote edge service (which consists in image processing for object recognition) under the constraints of energy, delay, and computing power. On the other hand, computation offloading is only possible when the drone is connected to the wireless link. Moreover, when the quality of the communication degrades, computation offloading may fail (resulting in retransmission), or can introduce an additional delay, which results in service degradetion and unnecessary power consumption.

Fig. 3 shows the prototypes of the implemented scenarios. As for the first scenario, at physical level, wearable sensors conneted to a ESP32 microcontrollers collect and stream patient vital signs to the gateway via WiFi. The edge server (the gateway) has been implemented by using Raspberry Pi 3 Model B+. The smart sensor is equipped with GSM/GPRS SIM808 modules (with GPS antennas for satellite localization).

In the second scenario, we assume that several smart nodes equipped with different sensors are deployed in each building environment (including advanced lighting systems, thermostats, meters, actuators, buzzer, display, etc.). As Fig. 3 shows, the smart sensors are implemented by ESP32 microcontrollers interconnected under a single

WLAN (Wireless Local-Area Network) exploiting the ESP-WIFI-MESH networking protocol built atop the Wi-Fi protocol [22]. ESP-WIFI-MESH is self-organizing and self-healing, meaning that the network can be built and maintained autonomously. The nodes are not required to connect to a central node. They are mutually responsible for relaying each others transmissions. This allows an ESP-WIFI-MESH network to have much greater coverage area as nodes can still achieve interconnectivity without needing to be in range of the access point.

For both scenarios, custom Python script have been written and deployed on each gataway in order to collect data from the smart devices, react to critical situation, and send data to a private cloud data-center. At cloud level, applications have been developed to collect data and format and display the alerts in user-friendly interfaces (with basics search&filtering capabilities).

As regarding the remote surveillance system, it consists of a Pixhawk that is the flight controller (the heart of the drone). It is a standard hardware for open-source autopilots. Moreover, the drone is equipped with a GPS antenna with Compass, a LiPo battery, a cam, and 6 motors. Two RFD900x modules (telemetry antenna) allow communication between the ground and on-board system. The OpenMV Cam microcontroller board is used to implement the machine vision application by the drone. The ground control station hosts the Mission Planner control software, which is used for plunning and control the mission. A TensorFlow-based application is deployed on the edge server for supporting image processing and object recognition services.

To implement and monitor the whole IoT applications, we relied on *DeviceHive*, an open source IoT platform providing instruments for smart devices communication and management. It is a scalable, hardware-agnostic microservice-based platform with device-management APIs for different protocols, which allows users to set up and monitor devices connectivity, control them and analyze their behaviour. DeviceHive provides REST, Web-Socket APIs plus MQTT API as an additional plugin. All communication is performed via JSON messages. Finally, we have implemented an observer to collect the considered metrics, which runs as a client of DeviceHive. It uses the MQTT protocol, which is tailored for communication in resource-constrained environments such as IoT. Message delivery in MQTT is done using publish-subscribe messaging protocol. Metrics are collected at data-center and MEC server/gateway levels, exploiting data made available from the Linux `proc` file system and, once collected, they are structured in a common JSON message and sent to the observer through an MQTT broker.

The hardware for deploying the testbed is as follows: A desktop PC (Intel i5 2400, 4GB of RAM), running an instance of DeviceHive Server, and collection scripts for metrics at data-center level; A laptop with Intel i5-3337U, 4GB of RAM, running an open source MQTT broker, called *mosquitto*, the open source version of InfluxDB server, and collection scripts for metrics at edge server/gateway level; Raspberry 3 Model B+ used as edge servers, running the edge services, and collection scripts for metrics at device level; and several ESP32 NodeMCU modules 2.4GHz WiFi + Bluetooth Dual Mode.

4.2 Security testing scenarios

For each of the three case studies, we have run eleven attacks at the data-center, edge server (referred to as *gateway* for brevity), and device level, selected to assess resilience under resource-exhaustion attack. They cover the most common categories of resource exhaustion attacks that afflict the three levels of the considered architecture: DoS, Flooding, and Brute-force attacks. Tab. 1 reports the full list, with the support tools used to implement them.

ID	Type of attack	Target	Support Tool
#1	HTTP DoS	Data-center	*Hulk* [25]
#2	TCP DoS	Data-center	*Sockstress* [26]
#3	Network Flooding	Data-center	*Hping3* [27]
#4	Slow POST DoS	Data-center	*ShowHTTPtest* [28]
#5	TCP/UDP DoS	Gateway	*LOIC* [29]
#6	Network Flooding	Gateway	*Hping3* [27]
#7	SSH Brute force	Gateway	*Hydra* [30]
#8	ICMP/HTTP DDoS	Gateway	*BoNeSi* [31]
#9	Bluetooth ping flooding	Device	*BlueSmack* [32]
#10	Bluetooth file transfer	Device	*Bluper* [33]
#11	WiFi Jamming	Device	*Wificurse* [34], [35]

TABLE 1: Attack scenarios

4.3 Metrics definition

Metrics are needed to characterize the system behavior from both system-level (i.e., resource usage) and user-level (i.e., performance) perspectives. The goal is to have metrics covering all partitions and layers of the architecture. As mentioned, layers are the data-center (DC), MEC server/gateway, and device; whereas, partitions are the subsystems within a given layer. For the data-center and MEC server/gateway, a partition is simply an OS subsystem as considered by Linux monitoring facilities (e.g., `top`, `ps`, `free`), namely, CPU, memory, I/O, and network. For the device layer, given their simplicity, we do not need to distinguish subsystems.

In order to characterize the subsystems of the data-center, the gateway, and device layers, as well as user-perceived performance, 364 metrics are collected. Tab. 2 lists the number of considered metrics for each layer. The main user-level performance metrics, measured on the control workload, are: *response time*, *throughput*, *longest/shortest transaction*, *concurrency degree*, and *latency*. In the UAV case study, due to its peculiarities, we also consider, in the evaluation of system performance and availability, uncertainty factors such as workload intensity and link reliability. The workload intensity is parametrized by the arrival rate of image processing requests, while the link reliability is considered as the communication disconnection rate. In fact, when the communication quality of drone (due to the attacks) becomes unstable, the service throughput degrades since many jobs cannot be successfully offloaded.

About data-center and gateway subsystems, Tab. 3 reports an excerpt of the most relevant metrics. While regarding the device-level, the collected metrics are: *power absorption*, *RSSI* (Received Signal Strength Indicator), *latency*, *response time*, and *used memory*.

4.4 Test execution and data collection

Testing sessions have a duration of 10 minutes for the experimental scenario 1 and 3. Since we also aim at investigating

Architectural layer	Number of metrics
Data-center and Gateway	9 *for CPU* 36 *for MEM* 13 *for IO* 287 *for NET*
Device	5
User-level performance	14
Total	364

TABLE 2: Metrics distribution.

the impact of longer attacks, we selected scenario 2 to run attacks with a much longer duration, namely 120 minutes. In fact, in this scenario we have a mesh network of sensors in the smart building, which is potentially more sensitive to smart sensors' battery depletion attacks conducted by jamming [35].[1]

In any case, the sample period is 5 seconds. The whole test is divided into three phases. A *pre-attack* phase that lasts 2.5 minutes (i.e., 30 samples), during which the system runs the workload to retrieve data from the smart sensors, querying them periodically each second. The *attack* phase, in which one attack from the list of Tab. 1 is run. This lasts for 5 minutes (i.e., 60 samples) for scenario 1 and 3, and 115 minutes for scenario 2. The *post-attack* phase, in which the attack is stopped and the system runs the same workload as the pre-attack stage, again for 2.5 minutes. The output of this process are time series X_i ($i = 1, \ldots, m$, with m being the number of metrics), each with a set of $n = 120$ (or $n = 1440$ for scenario 2) values $x_{i,t}$ (with $i = 1, \ldots, m$; $t = 1, \ldots, n$).

4.5 Data analysis

4.5.1 Preprocessing

Data cleaning and homogenization. First, we remove the time series with all values equal to a constant. Then, since all the metrics are conceived to represent the "consumption" of a resource, whenever the original metric has the opposite meaning (i.e., representing the availability of a resource, such as *free memory*), the values of the corresponding time series are multiplied by -1. Hence, an increase (decrease) in the values of a time series represents, for all the metrics, an increased (decreased) resource consumption.

Normalisation. All time series are normalized so as to avoid any bias due to different units of measures. For each time series X_i, we use the min-max normalization in order to have all metrics between 0 and 1:

$$x'_{i,t} = \frac{x_{i,t} - min_t\{x_{i,t}\}}{max_t\{x_{i,t}\} - min_i\{x_{i,t}\}}, \quad (1)$$

where $x'_{i,t}$ is the normalised value of $x_{i,t}$.

4.5.2 Impact analysis (SG1)

Resilience assessment. Resilience to resource exhaustion attacks is measured as a proper combination of the above metrics. The idea is to measure the resource depletion due to the attack. Our conjecture is that, in order to have a comprehensive and useful measure, we need to account for all the layers (and partitions) of the IoT architecture and for the user-perceived performance as well. Thus, we first focus on measuring attack-caused resource depletion in each subsystem, and then on combining them. In particular, we apply the following steps:

- For each partition p of each architectural layer l (i.e., subsystem $S_{p,l}$), and for the performance metrics as well, the gathered (normalized) time series are transformed in order to remove the (first-order) inter-correlation – being metrics of the same partition, they are typically highly correlated. This is done by principal component analysis (PCA), a common technique that transforms the original metric space into a space whose dimensions (i.e., the *principal components*) are linearly uncorrelated. Given a set of m metrics, a set of m uncorrelated principal components (PC) is obtained as output, in descending order of explained variance of the original dataset. We select a subset k of them in order to keep at least the 95% of the original variance and, at the same time, reduce dimensionality.
- The selected k PCs for $S_{p,l}$ are then combined into one time series used as a synthetic indicator of the trend, which captures the resource usage of that subsystem. This is done by Hotelling's multivariate control charts [36]. Multivariate control charts are used to monitor two or more interrelated process variables in order to detect shifts in the mean or the relationship between several interrelated variables. A Hotelling's chart computes a statistic that uniquely lends itself to plotting multivariate observations, named T^2, which combines the information from the dispersion and mean of several related variables. The T^2 computation considers two phases: in phase I, the observations of the system when it is not under attack are used to characterize the *normal operation mode*; in phase II, all the observations (including the attack) are used, and the T_i^2 value for the i^{th} observation of the k PCs is as follows:

$$T_i^2 = (Y_i - \bar{X})' S_X^{-1} (Y_i - \bar{X}), \quad (2)$$

where Y_i is the vector of k measurements for observation i, $\bar{X}$ is the vector of sample means of the k variables calculated from the dataset of phase I, S^{-1} is the inverse of the sample covariance matrix from the dataset of phase I, which provides information regarding the relationship between different variables. Values of the T^2 are compared to an upper control limit (UCL): points above the UCL are those values outside (with 95% of confidence) the normal operation mode range. The UCL is obtained as in [37]:

$$UCL = \frac{k(m+1)(m-1)}{m(m-k)} F_{1-\alpha,k,m-k}, \quad (3)$$

where m is the number of observations of phase I; α is the significance level ($\alpha = .05$ in our case) and F is the *Fisher* distribution. In our case, $m \leq 100$; if $m > 100$, the above formula changes. The output of this step is a $T_{p,l}^2$ time series, with the corresponding

1. Jamming, in fact, can compromise wireless connections through collisions. It can be implemented by broadcasting on the same frequency band as the sensors. During the attack, the smart sensors increase the transmission power to counteract noise and improve the transmission range and signal quality, at the expense of rapid battery depletion.

CPU	
Name	Description
User	Time spent running in user mode
System	Time spent in kernel mode
Idle	Time with CPU doing no work
Nice	Time with niced processes in user mode
IOwait	Time waiting for I/O to be completed
Softirq	Time to serve SW interrupts
Steal	Time stolen by other OSs in a virtual env.
Guest	Time spent for virtual CPU or Guest OS
...	...

NET	
Name	Description
IPInReceives	# received input datagrams
IPInHdrErrors	# input datagrams with header errors
ICMPInMsgs	# received ICMP messages
ICMPErrors	# received ICMP-specific errors
TCPIn/OutSegs	# segments received/sent
TCPRetransSegs	# segments retransmitted
UDPIn/OutDtg	# UDP datagrams to/from UDP users
UDPNoPorts	# UDP datagrams to closed dst port
...	...

MEM	
Name	Description
Free cached	Physical RAM used as cache
Buffers	Physical RAM used for file buffers
Used RSS	Physical RAM used as resident set size
Swap	Amount of swapped memory
Active/Inactive	Most/least recently used memory
Wrtback	Memory written back to the disk
Dirty	Non-file backed pages mapped
...	...

IO	
Name	Description
Reads completed	# reads successfully completed
Read sectors	# sectors read successfully
I/O in progress	# I/O ops. currently in progress
I/O in progress	# I/O ops. currently in progress
I/O time	Time spent with one I/O op. in progress
Weig. I/O Time	# I/O ops. in progress weighted by I/O time
%Util	%time to issue I/O requests to the device
...	...

TABLE 3: Excerpt from the list of metrics for data-center/gateway subsystems.

UCL value, for each subsystem $S_{p,l}$ (used for both average and point-wise *resilience score* described hereafter), plus a further T^2 time series for performance metrics. The T^2 time series represents a point-wise synthetic indicator of the metrics it is derived from.

The resilience score of each subsystem $S_{p,l}$, denoted as $R_{p,l} \in [-1; 1]$, is obtained by comparing the relative increase or decrease of the point-wise synthetic indicator (T^2) during the attack with respect to the average of pre- and post-attack. This synthetically captures the resource depletion experienced during the attack. Let us denote by $\bar{T^2}_{attack}$ and $\bar{T^2}_{idle}$ the average over time of the T^2 score respectively during the attack and during the pre- and post-attack phases (where we use median as average, since it is more robust to spikes). The resilience score for subsystem $S_{p,l}$ is defined as:

$$R_{p,l} = \begin{cases} -\frac{[\bar{T^2}_{attack} - \bar{T^2}_{idle}]}{\bar{T^2}_{attack}} & \text{if } \bar{T^2}_{attack} > \bar{T^2}_{idle}, \\ \frac{[\bar{T^2}_{attack} - \bar{T^2}_{idle}]}{\bar{T^2}_{idle}} & \text{if } \bar{T^2}_{attack} \leq \bar{T^2}_{idle}. \end{cases} \quad (4)$$

This gives a value $\in [-1; 0[$ whenever the attack produces an effect on that subsystem, causing the increase of $\bar{T^2}_{attack}$; while it gives a value $\in [0; 1]$ if the attack has no effect, thus $\bar{T^2}_{attack}$ is equal or even smaller than $\bar{T^2}_{idle}$. All the metrics, after preprocessing, represent resource consumption – hence an increase of their value (consequently of T^2) means an increase in resource consumption. In the above Equation, if $\bar{T^2}_{idle} = 0$ *and* $\bar{T^2}_{attack} = 0$, then $R_{p,l}$ is set to 0 (i.e., the attack does not affect resilience).

The same applies to the user performance T^2 time series, giving a corresponding resilience score R_{up}.

Average resilience score. For a summary metric, we average all partitions, each with its relative importance. Depending on the context, each partition can be considered equally important or not. For instance, one may consider that, at the device level, power consumption may be deemed more important than memory utilization, which is not true at MEC server/gateway or data-center level.

The $R_{p,l}$ scores are properly weighted according to the relative importance of each subsystem by applying an aggregation operator $op(\cdot)$. Denoting the set of weights $w_{p,l}$ assigned to each subsystem ($w_{p,l} \in [0, 1]$, $\sum_{p,l} w_{p,l} = 1$), resilience of a layer l is obtained as $R_l = op(w_{p,l}, R_{p,l}), \forall p$ and a fixed l; resilience of the whole system is $R = op(w_{p,l}, R_{p,l})$, $\forall p, \forall l$. We apply the *mean* as operator, obtaining:

$$R = \sum_{p,l} w_{p,l} R_{p,l}. \quad (5)$$

Point-wise resilience analysis. Each T^2 is a point-wise resiliency indicator. While the resilience score gives one aggregate measure, we consider the whole T^2 time series along with the UCL to have detailed figures of interest and explanations for a low or high resilience score of a subsystem. Indeed, the T^2 time series enables a point-wise analysis looking at the system behavior during the attack with respect to the normal operation mode (e.g., to understand the latency of reaction to an attack). One can look at the T^2 series of each subsystem (to analyze the behavior of every single subsystem), as well as at the aggregate T^2 series (e.g., obtained from the average of subsystem T^2 series). In both cases, given a T^2 series, the analyst can focus on the points that are out of statistical control, i.e., above the UCL, representing those samples in which the behavior is significantly different from the normal operation mode. The curve also helps to distinguish if a low resilience is due to few spikes exceeding the UCL by a large amount or to a

series of many points above the UCL by a small amount. Section 5 will show the results of this analysis.

4.5.3 Propagation analysis (SG2)

Propagation analysis aims at assessing to what extent the attacks on one layer of the MEC-IoT architecture have a perceivable effect on end users. To this aim, we analyze the causality relationships between the time series of multiple (sets of) metrics. Again, we consider the subsystems $S_{p,l}$, each with its own set of metrics. Let us denote the time series' vectors for the set of metrics for subsystem $S_{p,l}$ as $\mathbf{X}_{p,l}$ and the time series' vectors of user-perceived metrics as $\mathbf{Y}$. We look at the (multivariate) transfer entropy ($MuTE$) between every layers and user-level metrics, denoted as $MuTE(\mathbf{X}_{p,l}, \mathbf{Y})$. By considering the above time series $\mathbf{X}_{p,l}$ and $\mathbf{Y}$ as stochastic time-varying variables, the multivariate transfer entropy can be defined in terms of the associated conditional Shannon entropies ($H(\cdot|\cdot)$):

$$MuTE(\mathbf{X}_{p,l}, \mathbf{Y}) = H(\mathbf{Y}^t|\mathbf{Y}^{\bar{t}}) - H(\mathbf{Y}^t|\mathbf{Y}^{\bar{t}}, \mathbf{X}^{\bar{t}}_{p,l}), \tag{6}$$

where the superscript notation with t indicates the variables at time t while $\mathbf{Y}^{\bar{t}}$ are the vectors representing the entire past of the variable $\mathbf{Y}$ at times $t-1, t-2, \ldots$.

However, since the strong inter-correlation among the metrics of a subsystem can negatively impact the $MuTE$ result, we apply $MuTE$ using directly the T^2 indicators, as they represent an average synthetic indicator for each subsystem in which the first-order correlation has been removed using PCA: it becomes $MuTE(T^2_{p,l}, \mathbf{Y})$.

Intuitively, transfer entropy gives us the amount of information that a source variable tells us about a destination one, in the context of the destinations current state. Thus, it can be considered a good method to estimate the extent of the cause-effect relation between two time series: unlike correlation, it informs us about causality (it actually generalizes the Granger causality test) by computing how much information about the transition between two consecutive steps of the $\mathbf{Y}$ time series can be found in the past state of T^2. How "far" the past is, depends on the user-defined delay between the two time series d. We use delays going from 1 to 10 samples (e.g., 5 to 50 seconds). As we are interested in detecting potential transfers of information, we consider the maximum value across the 10 delay values. It is worth noting that results across the different delays are similar, with differences in the order of $10e$-3. The used TE estimator is the Kraskov-Stgbauer-Grassberger (KSG) estimator [38].[2] The higher its value, the stronger the cause-effect relationship between the source and the target time series. The output of this analysis allows one to assess, given an attack, which subsystem is the strongest cause of performance degradation experienced by end users.

4.5.4 Correlation analysis (SG3)

This step enables the investigation of what metrics are the most useful indicators to detect a resource exhaustion attack, e.g., by ML-based strategies. To identify the best set of metrics, feature selection methods are a viable option. Many methods can be used for this purpose – a common classification is based on the possible interaction with the learning model, categorizing the methods as: *filter*, *wrapper*, and *embedded* methods. Filter methods, such as Information gain, Chi-square test, Fisher score or Laplacian score, calculate the weight of each feature according only to the intrinsic properties of the features (distribution). Wrapper methods, such as recursive feature elimination, or sequential feature selection, select the best feature subset based on a specific classifier's results – therefore, although can give better results, its application and performance are dependent on the used classifier. In embedded methods, the feature selection process is done during the training of the classifier itself, namely feature selection is "embedded" in the construction of the classifier (examples are SVM, decision trees, and random forest), which can output feature weights.

Since the goal at this stage is not to build a classifier for attack detection (an attack detection algorithm needs to consider other factors besides features engineering [40]), filter methods are the best option, as they are computationally fast and independent of any downstream classifier that could be used at later stages. We use a well-known algorithm, the *Information Gain* (IG) attribute ranking [41], [42]; the algorithm evaluates the Information gain of each feature in the context of the target variable. In our case, we use all the gathered metrics as features $\mathbf{X}$ and, as target variable, a label Y (yes/no) for each collected sample denoting if the system is under attack or not at the time the sample is collected. On such a dataset, resulting from the execution of the above-described testing scenarios, we run IG and rate the metrics according to their potential contribution to the prediction. Specifically, IG ranks the attributes by their *information gain*, given by:

$$\begin{aligned} &IG(A_i) = H(C) - H(C|A_i) = \\ &= -\sum_{c \in C} P(c) \cdot \log_2 P(c) + \\ &- \left[-\textstyle\sum_{a \in D(A_i)} P(a) \sum_{c \in C} P(c|a) \cdot \log_2 P(c|a) \right], \end{aligned} \tag{7}$$

where $H(\cdot)$ denotes Shannon entropy, as in precedence, $C = \{Attack, \neg Attack\}$, $D(A_i)$ is the set of values of attribute A_i, and the output information gain measures the number of bits of information obtained by knowing the value of attribute A_i [42]. Numeric $D(A_i)$ values are discretized in 10 bins.

The output of this analysis is the set of most important metrics to explain the variation of the MEC-IoT system behavior under a specific attack (which are also the best ones to detect or predict an attack of that type).

5 Results

5.1 Resilience and impact analysis

Average resilience assessment. To compute the average score, we first define the weights assigned to each subsystem, $w_{p,l}$. We consider two options for illustrative purposes: the simplest option is to assign equal weight to each subsystem; the second option is to give different weights to different subsystems for each attack. In particular, for attacks directed

2. The tool used for computing TE is JIDT available at: https://github.com/jlizier/jidt

Layer	Subsystem1	Case Study	DC Attack #1	#2	#3	#4	Gateway Attack #5	#6	#7	#8	Device Attack #9	#10	#11	Mean
DC	CPU DC	1	-0,9993	-0,9981	-0,8840	-0,9994	-0,1426	0,0618	-0,8016	-0,4599	-0,7292	0,6845	-0,2004	**-0,4971**
		2	-0,9999	-0,9999	-0,9647	-0,9999	-0,1566	0,0595	-0,8666	-0,4902	-0,7747	0,6604	-0,2071	**-0,5218**
		3	-0,9999	-0,9999	-0,9325	-0,9999	-0,1416	0,0590	-0,8305	-0,4920	-0,7600	0,6473	-0,2115	**-0,5147**
	IO DC	1	-0,7802	-0,1733	-0,3069	0,0518	-0,0416	0,1957	0,1571	-0,5240	0,2734	0,0407	0,0563	**-0,0955**
		2	-0,8000	-0,1735	-0,3266	0,0512	-0,0426	0,1847	0,1431	-0,5269	0,2716	0,0381	0,0533	**-0,1025**
		3	-0,8271	-0,1804	-0,3210	0,0496	-0,0441	0,1972	0,1466	-0,5646	0,2549	0,0391	0,0534	**-0,1088**
	MEM DC	1	-0,9801	-0,9932	-0,9412	0,8760	0,1408	-0,8646	-0,9274	-0,9946	-0,6912	0,4576	0,0673	**-0,4410**
		2	-0,9999	-0,9999	-0,9999	0,8220	0,1330	-0,9179	-0,9999	-0,9999	-0,7382	0,4558	0,0657	**-0,4708**
		3	-0,9999	-0,9988	-0,9822	0,8472	0,1422	-0,9383	-0,9325	-0,9999	-0,7468	0,4316	0,0650	**-0,4648**
	NET DC	1	-0,2210	-0,6941	0,0167	0,7654	-0,3426	-0,1838	0,0761	-0,3599	-0,2567	-0,2374	-0,2290	**-0,1515**
		2	-0,2260	-0,7488	0,0160	0,7531	-0,3396	-0,1872	0,0708	-0,3888	-0,2742	-0,2485	-0,2331	**-0,1642**
		3	-0,2250	-0,7195	0,0167	0,6928	-0,3558	-0,1845	0,0704	-0,3955	-0,2648	-0,2443	-0,2288	**-0,1671**
Gateway	CPU Gtw	1	-0,8534	-0,8639	0,0137	-0,0019	-0,0390	-0,8009	-0,9989	-0,9612	-0,6329	-0,5696	-0,4041	**-0,5556**
		2	-0,9181	-0,8883	0,0128	-0,0020	-0,0408	-0,8177	-0,9999	-0,9754	-0,6340	-0,5734	-0,4122	**-0,5681**
		3	-0,8451	-0,9464	0,0129	-0,0020	-0,0426	-0,8672	-0,9999	-0,9999	-0,6408	-0,5647	-0,4072	**-0,5730**
	IO Gtw	1	0,1068	-0,0114	-0,1923	0,0969	0,4264	0,1091	-0,1239	-0,2226	-0,0511	-0,1169	-0,0466	**-0,0023**
		2	0,1062	-0,0113	-0,2077	0,0884	0,4198	0,0997	-0,1307	-0,2303	-0,0511	-0,1188	-0,0505	**-0,0078**
		3	0,1075	-0,0119	-0,1935	0,0947	0,3950	0,1087	-0,1279	-0,2323	-0,0534	-0,1196	-0,0492	**-0,0074**
	MEM Gtw	1	-0,7709	-0,8344	-0,8821	0,8539	0,0910	-0,9252	-0,9999	0,0956	-0,3476	-0,2818	-0,2632	**-0,3877**
		2	-0,8025	-0,8755	-0,9506	0,7881	0,0909	-0,9999	-0,9999	0,0956	-0,3530	-0,3095	-0,2870	**-0,4185**
		3	-0,7717	-0,8784	-0,8923	0,8414	0,0857	-0,9186	-0,9999	0,0960	-0,3647	-0,2971	-0,2745	**-0,3976**
	NET Gtw	1	-0,3098	-0,6627	0,4823	0,0432	-0,2111	-0,1904	-0,3116	0,1313	-0,1523	-0,2435	-0,1855	**-0,1464**
		2	-0,3363	-0,6856	0,4574	0,0407	-0,2233	-0,2004	-0,3222	0,1187	-0,1596	-0,2436	-0,1859	**-0,1582**
		3	-0,3232	-0,7285	0,4709	0,0427	-0,2306	-0,2023	-0,3181	0,1270	-0,1641	-0,2601	-0,2015	**-0,1625**
Device	DEVICE	1	0,3103	-0,0164	-0,2514	-0,0482	0,1793	0,0390	0,1422	0,2810	-0,9925	-0,9689	-0,9622	**-0,2080**
		2	0,2926	-0,0174	-0,2864	-0,0491	0,1792	0,0374	0,1359	0,2651	-0,9999	-0,9999	-0,9999	**-0,2220**
		3	0,3038	-0,0180	-0,2602	-0,0492	0,1682	0,0365	0,1342	0,2694	-0,9934	-0,9794	-0,9999	**-0,2171**

			#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	Mean
System avg	System Un.	1	-0,4997	-0,5830	-0,3272	0,1819	0,0067	-0,2843	-0,4208	-0,3349	-0,3978	-0,1372	-0,2408	**-3,0371**
		2	-0,5204	-0,6000	-0,3611	0,1658	0,0022	-0,3046	-0,4410	-0,3480	-0,4126	-0,1488	-0,2507	**-3,2193**
		3	-0,5090	-0,6091	-0,3424	0,1686	-0,0026	-0,3011	-0,4286	-0,3546	-0,4148	-0,1497	-0,2505	**-3,1937**
	System W.	1	-0,5752	-0,6235	-0,3892	0,1793	-0,0250	-0,3359	-0,4786	-0,3054	-0,4572	-0,2204	-0,3129	**-3,5440**
		2	-0,5931	-0,6402	-0,4250	0,1630	0,0205	-0,3585	-0,4940	-0,3172	-0,4713	-0,2339	-0,3257	**-3,6753**
		3	-0,5871	-0,6446	-0,4077	0,1621	0,0141	-0,3530	-0,4849	-0,3232	-0,4727	-0,2327	-0,3254	**-3,6550**
User Perf.	User Perf.	1	-0,9859	-0,9994	-0,2541	-0,9329	-0,0998	-0,1437	-0,7267	-0,2136	-0,7993	-0,6351	-0,6522	**-6,4427**
		2	-0,9936	-0,9999	-0,2571	-0,9242	-0,1082	-0,1452	-0,7556	-0,2197	-0,7917	-0,6655	-0,6633	**-6,5240**
		3	-0,9999	-0,9999	-0,2645	-0,9871	-0,1066	-0,1509	-0,7887	-0,2506	-0,8816	-0,6305	-0,7620	**-6,8223**

Less resilient — More resilient

TABLE 4: Resilience score for each attack, for the three case studies.

to a given layer, we have assigned a weight to the subsystems of that layer that is double the weights assigned to the others (ensuring $\sum_{p,l} w_{p,l} = 1$).

Tab. 4 reports the resilience score for each attack scenario, showing both the partial score $R_{p,l}$ for each subsystem $S_{p,l}$ and the global score for the two weights assignment options, denoted as $System_U$ and $System_W$, respectively (U: uniform; W: weighted). The user performance score is also reported in the last row.

In case study 1, we notice that the most impacted subsystems in the first four attacks (i.e., data-center level attacks) are the CPU and memory of the host nodes. In these attacks, the CPU and especially the memory of the gateway are also influenced, as the load pressure on the data-center reflects on the gateway it interacts with. Gateway-level attacks (#5 to #8) also heavily impact the gateway CPU and memory (attack #8 does not impact the memory; attack #5 is not impactful); the CPU of the data-center seems to be not greatly impacted (except for attack #7), while the memory of data-center experiences a depletion too in attacks #6 to #8. The other subsystems are not impacted. Device-level attacks (#9 to #11) hits exclusively the device, with marginal impact on the gateway and the data-center (more in attack #9). Finally, user performance is impacted mainly by attacks at the data-center and at the device level as well. However, it is worth noting that user performances are not always impacted by the attack (e.g., attack #5, #6, and #8 have a small impact, although there are subsystems heavily affected by attack #6 and #8). Thus, the attacks do not always propagate to the user level in the time interval that we considered. This will be clearer in the propagation analysis study.

In case study 2, we notice differences in the absolute values of resilience scores, since the longer duration leads resources to exhaust more severely. The last column reports the mean over the attacks for the three case studies, and case study 2 has consistently worse resiliency scores than case study 1. It is worth to notice the extremely low resilience score at device level in device-directed attacks (number 9, 10 and 11), $R = -0.9999$ in all the three cases, especially due to battery depletion of the smart sensors involved. The point-wise indicator in the next paragraph highlights the differences in the degradation dynamics that we observed.

In case study 3, the differences are in the user-perceived metrics: in this case, for instance, attack number 9 at device level caused a high communication disconnection rate, in turn leading to a low image processing rate, resulting in a significantly lower resilience at user-level (i.e., on user perceived service metrics) compared to scenario 1 and 2.

Therefore, resilience score of user-performance metrics in attack 9 for case study 3 is -0.8816 compared to -0.7993 and -0.7917 of case 1 and 2.

Point-wise resilience analysis. Figures 4a-4k report an example of T^2 for each attack in case study 1, where we reported the T^2 of the subsystem with the smallest resilience score (as per Tab. 4), namely the ones more heavily affected by the attack. It should be recalled that the resilience score is negative when the T^2 value under attack (i.e., in the central part of the graph) is greater than the T^2 in pre-post attack phase (i.e., at the extremes). These graphs show the type of output the analyst will get; the indicators provide hints on the pattern of resource depletion caused by the attack. For instance, in all cases, it can be noticed the reaction of the system to the attack. In almost all the cases the attack consistently causes the degradation of resources for all its duration, with many violations of the UCL representing the normal behavior; in a few cases (e.g., attack #3 and #8) few spikes cause the resilience score to be particularly small.

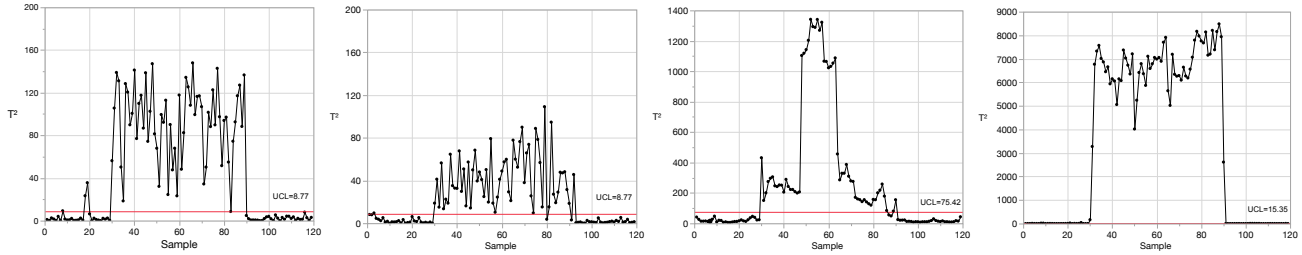
While case study 3 exhibits very similar patterns, although more pronounced on the user-level metrics, it is interesting to investigate the behaviour in case study 2, wherein we investigated the behaviour over a longer duration of the attack. In this case, the longer duration (2 hours) makes it interesting to look at possible trends in the resource consumption – which entails a trend in the T^2 value denoting a progressively worse resilience over time. Monitoring such trends could support mitigation strategies before the failure, such as capping policies to limit the workload or proactive maintenance actions such as rejuvenation [50]. Figures 5a – 5c report three representative examples, one per monitored subsystem (DC, gateway and device), obtained under three attacks. As explained in Section 4.5.1, the increasing trend of T^2 denotes an increasing resource consumption for the subsystem under analysis. However, T^2 is a combination of all the normalized metric values of the subsystem and has not an immediate physical interpretation: if we want to figure out which particular resource is being depleted more quickly, we need to look at the individual metrics. We have therefore estimated the trends' slopes of the most critical metric (i.e., the one referring to the resource that is depleting more quickly) and computed the so-called *time-to-exhaustion*, which is the total time needed to exhaust the resource. The three metrics turned out to be the *free swap space*, the *physical memory available*, and the device's battery (i.e.,*power consumption*) for, respectively, the three cases of Figure 5a – 5c.

In particular, in order to assess the extent of the trend, we use the non-parametric Mann-Kendall test [51] to detect if the trend is statistically significant or not, with significance level $\alpha = .05$, followed by the non-parametric Theil-Sen's procedure [52] to estimate the slope of the trend. The TTE is computed as the initial total availability of the resources divided by the consumption trend: for DC, the initially available memory is 3,738,504 KB (the swap space is 4,342,484 KB); for gateway, the initially available memory is 949,444 KB (swap space 102,396 KB); for device, the initially available memory is 327.680 KB (and the power capacity is 1,700 mAh). The results are reported in the captions of Figures 5d-5f. For instance, in attack number 1 (a DC-directed attack), the estimated slope for the available swap space is 20,530 KB/min (i.e.: ≈20.5MB/min), thus: $TTE = (4,342,484/20,530) = 211.5$ minutes, which is 3.5 hours (namely, after 2 hours of test, approximately 1.5 hours remain before completely exhausting the swap space). If we consider that at time 0 the swap space could be not completely free, then this time is even shorter: for instance, in Figure-5d, the initial free space is approximately 3.5GB, which gives a $TTE = 2.8$ hours, hence 0.8 hours remain before exhaustion. In attack 7 (a gateway-directed attack), the TTE tells that a complete memory depletion of the gateway will occur in approximately 5 hours, while in attack 10 (a device-directed attack) the battery will exhaust in approximately 9.78 hours.[3] Clearly, the visual inspection remains fundamental, as the possible presence of non-linear trends would make the linear Sen's estimation too optimistic or pessimistic. More complex techniques for non-linear trend estimation can be implemented [53], [54], but are out of the scope of this work.

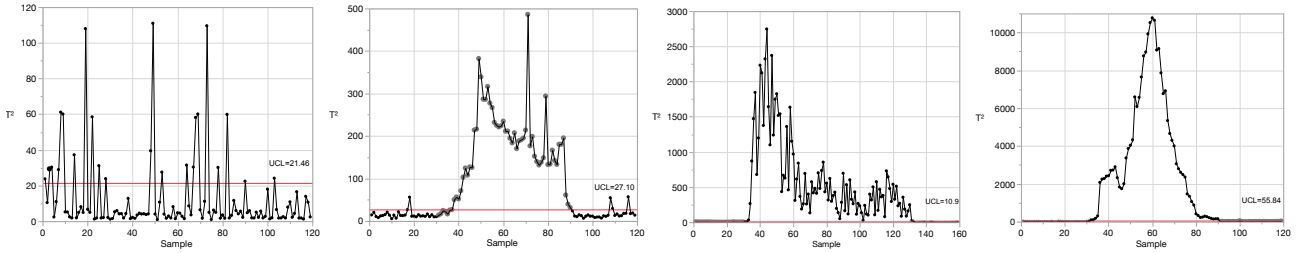
5.2 Propagation

Fig. 6 reports the transfer entropy from each subsystem T^2 time series to the *performance* T^2 time series in all the attacks. It represents the average information (measured in *nat*) from the subsystem's time series that helps predict the next values of performance in the context of its past. One *nat* is equal to $\frac{1}{ln2}$ bits. For instance, in the case of attack #1 in case study 1, the subsystem that "transfers" more information content to the performance time series is the CPU of the data-center. This is in line with the expectation: attack #1 acts on the data-center and the resilience score of its CPU subsystem is high compared to the others (cf. with Tab. 4). But this is not always the case, because the propagation also depends on the interactions between the subsystems and on the type of attack. For instance, in device-level attacks, we see lower absolute values of transfer entropy, because the propagation is across multiple layers before reaching the user interface. Hence, the effect of the attack may be mitigated (or at least "modified") by the gateway and data-center layer. In case study 1, when looking at the three types of attack, it is clear that the CPU usage at the data-center level reaches the highest value in DC attacks, even if the network subsystem plays a role too. In Gateway attacks, the CPU of the gateway is the one more causally related to user-level performance, but the IO and network of the gateway also appear as influential, and the data-center memory also reaches high values confirming results from Tab. 4. In device attacks, the values of device-level metrics are quite higher compared to the same metrics in DC and Gateway attack scenarios. Depending on the attack, proper protection/tolerance means can be placed in the most impactful layers by looking together at the results of Tab. 4 and Fig. 6.

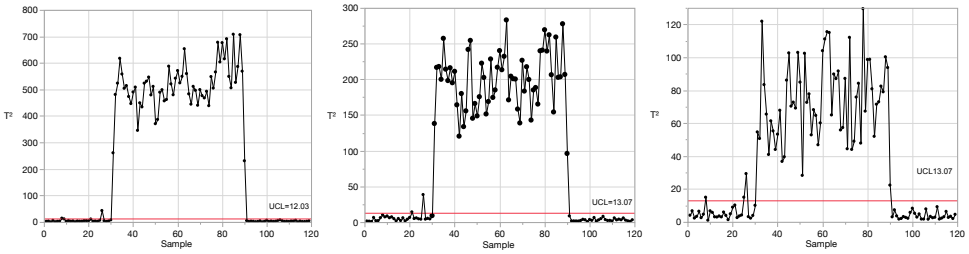
3. Note that, in the case of power consumption, the TTE computation needs to consider the relation between power consumption (in mA) and battery capacity (in mAh). In fact, in the first hour the average power consumption is 95.94mA, hence at that pace the battery would deplete in 1700mAh/95.95mA = 17 hours; however, the consumption increases with time, and the TTE is the x value such that the integration of power consumption over $[0, x]$ (divided by 60 minutes) equals 1700 mAh. This value is $TTE = 586.8$ minutes, i.e. 9.78 hours.



(a) Attack 1: CPU_{DC} subsystem (b) Attack 2: CPU_{DC} subsystem (c) Attack 3: MEM_{DC} subsystem (d) Attack 4: CPU_{DC} subsystem



(e) Attack 5: NET_{DC} subsystem (f) Attack 6: MEM_G subsystem (g) Attack 7: MEM_G subsystem (h) Attack 8: MEM_{DC} subsystem



(i) Attack 9: $DEVICE$ (j) Attack 10: $DEVICE$ (k) Attack 11: $DEVICE$

Fig. 4: Point-wise resilience indicator, reporting the subsystem affected more by the attack. UCL: *Upper Confidence Limit*, denoted by the red line

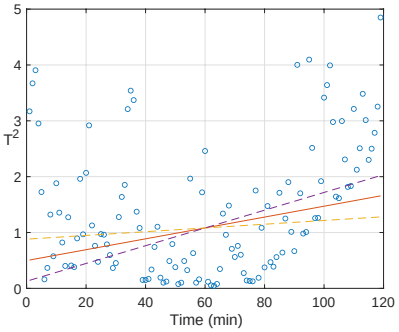
In case studies 2 and 3, patterns are slightly different, because of the network of sensors (case study 2) and a different communication structure (case study 3). For instance, in case study 3, attacks 9, 10 and 11 exhibit a higher transfer entropy between the device layer and the final user-perceived performance. This was also indirectly suggested by the worse average resilience score of user-performance metrics in the device-directed attacks (Table 4). In other words, the impact of the attacks at device level in case study 3 impacted more the user-perceived performance, likely because of the impact of the attack on the communication link between the UAV and the rest of the system, which is less reliable than the other two cases.

5.3 Correlation

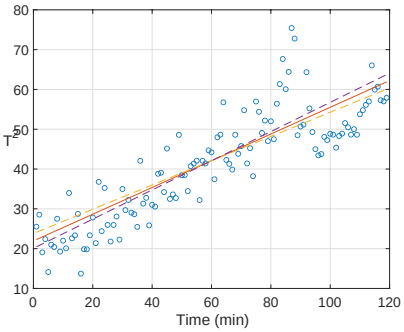
In this step, we are interested in understanding if some metrics could be more important than others in a possible prediction algorithm. By applying the IG algorithm [41], [42], we have the list of attributes ranked by their information gain. For each of the 11 attack scenario in each case study, we consider the top-10 ranked metrics list, and count the number of occurrences of a metric in these 11 lists (Fig. 7). Note that, the subsystems with the highest impact (e.g., data-center memory or CPU) do not necessarily have the best predictive ability. A subsystem can heavily be impacted by the attack, but the values of its metrics can be less related (in terms of information gain) to the binary outcome (attack = YES/NO) than a less impacted subsystem (as can be seen from Fig. 4a-4k). In general, we observe that memory-related metrics are very important as they appear across different subsystems (e.g., in an attack at the gateway level, there are data-center memory metrics in the top list). There are differences between the case studies. In case studies 2 and 3, the attacks at device level are more impactful (as evident from Figure 4 and Figure 6); in this case, the corresponding device-level metrics are also more related to the attack, unlike case study 1 – several device-level metrics appear at *device* layer of Figure 7 for case study 2 and 3. This is explained by the higher complexity of scenarios 2 and by the communication infrastructure of scenario 3 as mentioned above.

6 Related Work

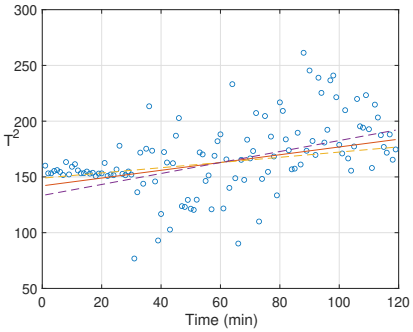
In [2], [14], it has been described how the security and resilience issues in the context of extended cloud computing can impact both MEC- and fog-based IoT systems. They examined the technologies that implement such models and architectures, as well as analyzed how security and resilience-related mechanisms applied in the cloud could be adapted in order to improve security and resilience in



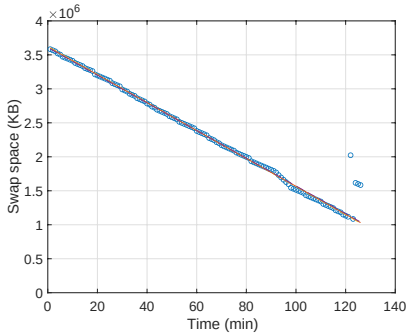
(a) T^2 of *DC* memory subsystem in Attack 1.



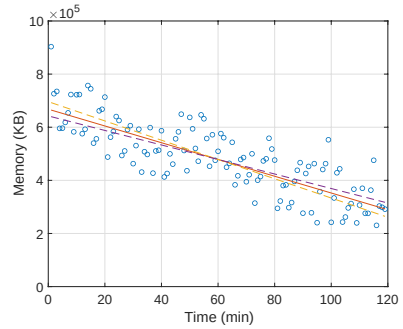
(b) T^2 of *gateway* subsystem in Attack 7.



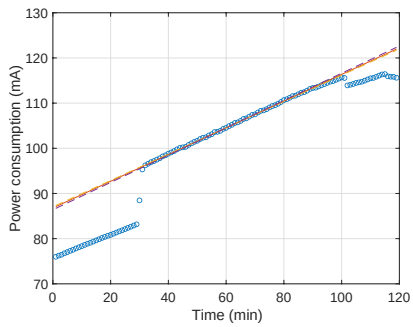
(c) T^2 of *device* subsystem in Attack 10.



(d) Free swap space of DC in attack 1. Trend: $-20,530\ KB/min$, $TTE = 211.5\ min$ (i.e.: 3.5 h)



(e) Physical memory available of gateway in attack 7. Trend: $-3,154.9\ KB/min$,$TTE = 300.9\ min$ (i.e.: 5.0 h)



(f) Power consumption of device in attack 10. Trend: $0.2957\ mA/min$,$TTE = 586.8\ min$ (i.e.: 9.78 h)

Fig. 5: Case study 2. Trend estimation for critical indicators, and point-wise resilience indicator T^2.

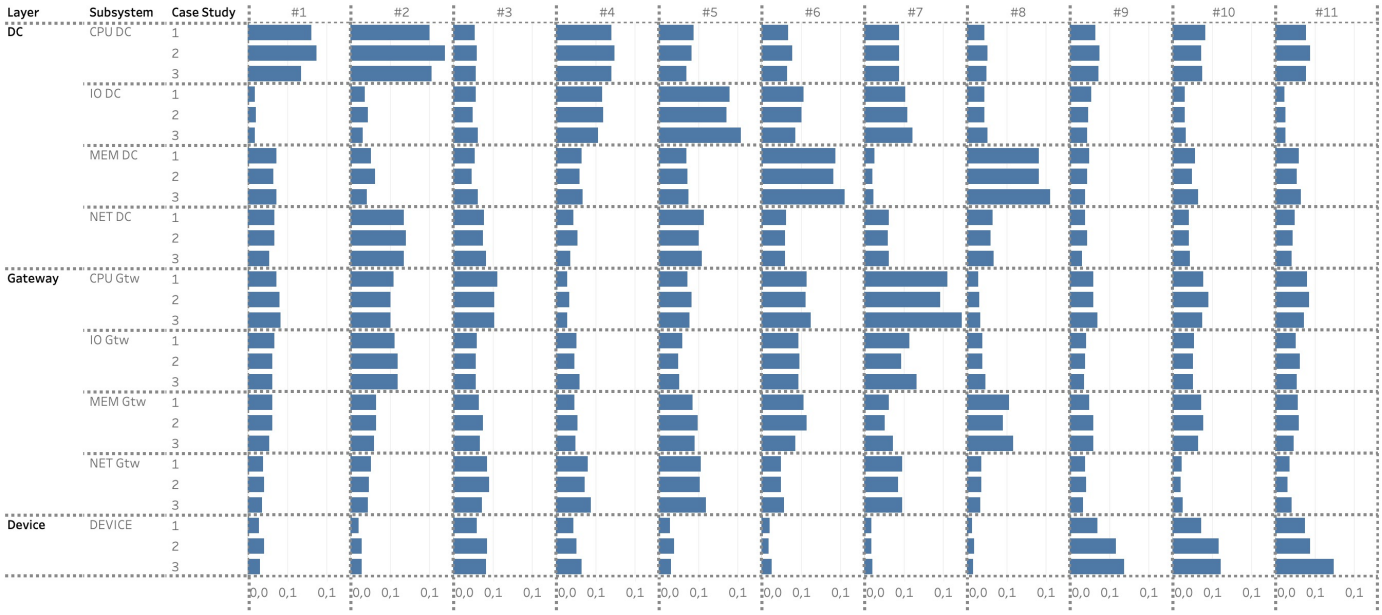


Fig. 6: Transfer entropy values, measured in *Nats*, and ranking

extended cloud environments. Although, in the context of a MEC node, exhausting attacks have limited scope since they affect only the attacked vicinity and not the entire network [16], [43], several works presented in the literature proposed schemas for fragmented dynamic deployments of defending mechanisms capable of mitigating the DoS attacks. In particular, to secure MEC-based IoT infrastructures and applications against DoS attacks, researchers proposed techniques based on network function virtualization (NFV) and software-defined network (SDN) to dynamically combine defense resources on demand. Mamolar et al. [44] presented an anti-DoS scheme, which leveraged NFV to enhance the deployment flexibility of intrusion detection systems in MEC scenarios. Other schemes have been proposed

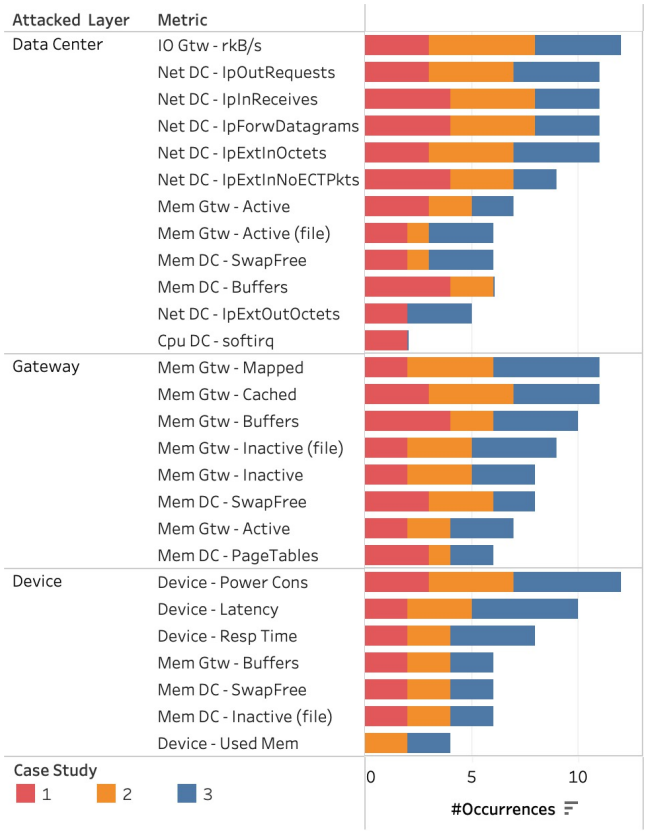


Fig. 7: List of metrics that occurred more than once in the top-10 metric lists

to dynamically allocate virtual defense resources depending on the attack intensity variation so as to balance the attack load [45], [46]. Another approach consists in dynamically establishing cooperation of defense resources. For example, Li. et al. [47] proposed a framework, which orchestrated virtualized intrusion protection system instances through SDN. MEC nodes can request additional defense resources from their directly connected neighbors. Tan et al. [48] extended this approach by proposing a global orchestration, which also involved the indirectly connected MEC nodes in order to increase defense cooperation. ResumeNet project [49] proposed a mobile edge platform that adopted several resilience principles and a strategy to detect, diagnose and recover from security attacks against the extended cloud. A policy engine was responsible for mapping detection events to reconfiguration actions on the base of a collection of policies. These works mainly focused on mechanisms and defense schemes to detect and mitigate DoS attacks. Our focus is on investigating how various types of resource exhaustion attacks impact the ability of a MEC-based IoT application in delivering an acceptable level of service. In particular, we have *i)* studied the resilience to attacks against all the layers (and partitions) of the architecture, *ii)* investigated the possible cause-effect relationship between the user-experienced performance degradation and the resource exhaustion under attack at all the layers/partitions, *iii)* analyzed the correlation between metrics at layer/partition level and the attacks.

7 Conclusions

In this work, we present a testing methodology to evaluate the impact of resource-exhausting attacks on an IoT application that exploits the services offered by a MEC server. The number of IoT application scenarios where the MEC paradigm can be applied is huge, and the effects that a successful attack might cause in such scenarios can be considerably harmful. The adopted methodology enables the analysts to study the resilience to such attacks as well as to support the root cause analysis by investigating the propagation of the effects across layers, hence supporting the consequent planning of proper fault tolerance means at the layer/partition level. Moreover, the correlation analysis can support the development of attack detection strategies in such architectures, based on those metrics that can contribute more to a downstream attack detection task.

References

[1] M. Weyrich and C. Ebert. Reference architectures for the internet of things, in *IEEE Software*, vol. 33, no. 1, Dec. 2015, pp. 112-116.

[2] S.N. Shirazi, A. Gouglidis, A. Farshad, D. Hutchison. The Extended Cloud: Review and Analysis of Mobile Edge Computing and Fog From a Security and Resilience Perspective, in *IEEE Jounal on Selected Areas in Communications*, vol. 35, no. 11, 2017, pp. 2586-2595.

[3] J.P. Sterbenz, D. Hutchison, E.K. etinkay, A. Jabbar, J.P. Rohrer, M. Schller, and P. Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines, in *Computer Network*, vol. 54, no. 8, 2010, pp. 12451265.

[4] J. Shropshire, Extending the cloud with fog: Security challenges & opportunities, in Proc. of the *20th Americas Conference on Information Systems*, 2014, pp. 110.

[5] S. Kounev, P. Reinecke, F. Brosig, J.T. Bradley, K. Joshi, V. Babka, A. Stefanek, and S. Gilmore. Providing dependability and resilience in the cloud: Challenges and opportunities, in *Resilience Assessment and Evaluation of Computing Systems*, Springer, May 2012, pp. 6581.

[6] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie. Mobile Edge Computing: A Survey, in *IEEE Internet of Things Journal*, vol. 5, no. 1, Feb. 2018, pp. 450-465.

[7] ETSI, Mobile-Edge Computing Introductory Technical White Paper, 2014, available at: http://www.etsi.org/technologies-clusters/technologies/mobile-edgecomputing [Last acc. May 2019].

[8] Y.C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young. Mobile Edge Computing: A key technology towards 5G, 2015, available at: http://www.etsi.org/technologiesclusters/technologies/mobile-edge-computing [Last accessed Apr. 2018].

[9] T.W. Nowak, M. Sepczuk, Z. Kotulski, W. Niewolski, R. Artych, K. Bocianiak, T. Osko, J.P. Wary. Verticals in 5G MEC-Use Cases and Security Challenges, in *IEEE Access*, vol. 9, 2021, pp. 87251-87298.

[10] N. Hehenkamp, C. Facchi, and S. Neumeier. How to achieve traffic safety with LTE and edge computing, in Proc. of *Advances in Information and Communication*, LNNS, vol. 69, 2020, pp. 164-176.

[11] S. Cheruvu, A. Kumar, N. Smith, and D. M. Wheeler. IoT vertical applications and associated security requirements, in Proc. of *Demystifying Internet of Things Security Successful IoT Device/Edge and Platform Security Deployment*, 2020, ch. 6.

[12] D. Miessler. Securing the Internet of things: Mapping IoT attack surface areas with the OWASP IoT top 10 project, in Proc. of the *RSA Conference*, 2015.

[13] R. Kozik, M. Chora, M. Ficco, and F. Palmieri. A scalable distributed machine learning approach for attack detection in edge computing environments, in *Journal of Parallel and Distributed Computing*, vol. 119, Sep. 2018, pp. 18-26.

[14] P. Ranaweera, A.D. Jurcut, and M. Liyanage. Survey on Multi-Access Edge Computing Security and Privacy, in *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, 2021, pp. 1079-1124.

[15] R. Romana, J. Lopez, M. Mambob. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges, in *Future Generation Computer Systems*, vol. 78, no. 2, 2018, pp. 680-698.

[16] P. Krishnan, S. Duttagupta, and K. Achuthan. SDNFV based threat monitoring and security framework for multi-access edge computing infrastructure, in *Mobile Netw. Appl.*, vol. 24, no. 6, Dec. 2019, pp. 1896-1923.

[17] J.F. Castet and J. H. Saleh. Survivability and Resiliency of Spacecraft and Space-Based Networks: a Framework for Characterization and Analysis, in *AIAA SPACE Conference*, 2008.
[18] P. Smith, D. Hutchison, J.P. Sterbenz, M. Scholler, A. Fessi, C. Lac, and B. Plattner. Network resilience: a systematic approach, in *IEEE Communications Magazine*, vol. 49, no. 7, 2011, pp. 88-97.
[19] D. Henry and J. E. Ramirez-Marquez. Generic metrics and quantitative approaches for system resilience as a function of time, in *Reliability Engineering & System Safety*, vol. 99, 2012, pp. 114-122.
[20] J.C. Laprie. Resilience for the scalability of dependability, in Proc. of the *4th IEEE Int. Symp. on Network Computing and Applications (NCA)*, 2005, pp. 5-6.
[21] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. Open-Flow: Meeting carrier-grade recovery requirements, in *Computer Communications*, vol. 36, no. 6, 2013, pp. 656-665.
[22] ESP-WIFI-MESH networking protocol, available at: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/esp-wifi-mesh.html [last access: Nov. 2023].
[23] M. Ficco, R. Palmiero, M. Rak, and D. Granata. MAVLink Protocol for Unmanned Aerial Vehicle: Vulnerabilities Analysis, in Proc. of the *IEEE Int. Conf. on Dependable, Autonomic and Secure Computing (DASC/PiCom/CBDCom/CyberSciTech)*, 2022, pp. 1-6.
[24] M. Ficco and M. Rak. Stealthy denial of service strategy in cloud computing, in *IIEEE Trans. Cloud Computing*, vol. 3, 2015, pp. 80-94.
[25] Hulk - HTTP DoS attack tool, available at: https://github.com/grafov/hulk [last access: Mar. 2020].
[26] Sockstress - TCP DoS attack tool, available at: https://github.com/defuse/sockstress [last access: Feb. 2020].
[27] Hping3 - Network flood DDoS, available at: https://linuxhint.com/hping3/ [last access: Mar. 2021].
[28] SlowHTTPtest - Slow HTTP POST DoS attack tool, available at: https://tools.kali.org/stress-testing/slowhttptest [last access: Mar. 2020].
[29] LOIC - TCP/UDP DoS attack tool, available at: https://sourceforge.net/projects/loic/ [last access: Mar. 2020].
[30] Hydra - SSH Brute force attack tool, available at: https://linuxconfig.org/ssh-password-testing-with-hydra-on-kali-linux [last access: Apr. 2020].
[31] BoNeSi - ICMP/HTTP DDoS attack tool, available at: https://github.com/Markus-Go/bonesi [last access: Gen. 2020].
[32] BlueSmack - Bluetooth ping flooding attack tool, available at: http://www.bluejackingtools.com/unix/bluesmack/ [last access: Feb. 2020].
[33] Bluper - Bluetooth file transfer, available at: https://downloadcenter.intel.com/it/download/30326?v=t [last access: Feb. 2020].
[34] Wificurse - WiFi Jamming attack tool, available at: https://github.com/oblique/wificurse [last access: Gen. 2020].
[35] WiFi-Jammer, available at: rhttps://github.com/DarkSyntax7/Wifi-Jammer
[36] N.D. Tracy, J.C. Young, and R.L. Mason. Multivariate Control Charts for Individual Observations, in *Journal of Quality Technology*, vol. 24, 1992, pp. 88-95.
[37] D.C. Montgomery. Introduction to Statistical Quality Control. 7th ed. New York: John Wiley & Sons. 2013.
[38] A. Kraskov, H. Stgbauer, and P. Grassberger. Estimating mutual information. Phys. Rev. E 69, 2004, Issue 6, pp. 1-16.
[39] N.A. M.R. Senaviratna and T.M. J.A. Cooray. Diagnosing Multicollinearity of Logistic Regression Model, in *Asian Journal of Probability and Statistics*, vol. 5, no. 2, Oct. 2019, pp. 1–9.
[40] Liu, H.; Lang, B. Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. In *Applied Sciences* 2019; 9 (20): 4396.
[41] M. Hall, G.Holmes. Benchmarking attribute selection techniques for discrete class data mining, in *IEEE Transactions on Knowledge and Data Engineering*, vol 15 , no. 6, 2003, pp. 1437-1447.
[42] Y. Yang, J. O. Pedersen. A comparative study on feature selection in text categorization, in *Proc. 14th Intl. Conf. on Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, pp. 412-420.
[43] R. Roman, J. Lopez, and M. Mambo. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges, in *Future Generation Computer Systems*, vol. 78, no. 2, Jan. 2018, pp. 680-698.
[44] A.S. Mamolar, Z. Pervez, Jose M. Alcaraz Calero, and Asad M. Khattak. Towards the transversal detection of ddos network attacks in 5g multi-tenant overlay networks, in *Computers & Security*, vol. 79, 2018, pp. 132147.
[45] K. Bhardwaj, J. C. Miranda, and A. Gavrilovska. Towards IoT-DDoS prevention using edge computing, in Proc. of the *24th USENIX Security Symposium*, 2018, pp. 171184.
[46] D. Sattar, A. Matrawy. Towards secure slicing: Using slice isolation to mitigate DDoS attacks on 5G core network slices, in Proc. of the *IEEE Conf. Commun. Netw. Security (CNS)*, 2019, pp. 8290.
[47] H. Li, L. Wang, Online orchestration of cooperative defense against ddos attacks for 5g mec, in Proc. the *15th IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2018, pp. 16.
[48] X. Tan, H. Li, L. Wang, and Z. Xu. Global Orchestration of Cooperative Defense against DDoS Attacks for MEC, in Proc. of the *15th IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1-6.
[49] $D^2R^2 + DR$ project, available at: http://www.comp.lancs.ac.uk/resilience/ [last access Feb. 2020].
[50] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. 2014. A survey of software aging and rejuvenation studies. J. Emerg. Technol. Comput. Syst. 10, 1, Article 8 (January 2014), 34 pages. https://doi.org/10.1145/2539117
[51] Henry B Mann. Nonparametric tests against trend, in *Econometrica: Journal of the econometric society*, 1945, pp. 245-259.
[52] P. J. Sen. Estimates of the regression coefficient based on Kendall's tau, in *Journal of the American statistical association*, Taylor & Francis Group, vol. 63, no. 324, 1968, pp. 1379-1389.
[53] G. Hoffmann, K. Trivedi, and M. Malek. A best practice guide to resource forecasting for computing systems. IEEE Transactions on Reliability, 56(4), 615628, 2007.
[54] P. Zheng, Y. Qi, Y. Zhou, P. Chen, J. Zhan, and M. R. T. Lyu. An automatic framework for detecting and characterizing performance degradation of software systems. IEEE Transactions on Reliability, 63(4), 927943, 2014.

Massimo Ficco is an Full Professor at the University of Salerno, Italy. His research interests include cloud and edge security, malware analysis, and software reliability of critical infrastructures. He has a Ph.D. in computer engineering from the University of Naples "Parthenope, Italy. He serves as the editor-in-chief, board-editor and reviewer of several international journals, and has been conference chair and member of many international conference committees.

Roberto Pietrantuono is Associate Professor at University of Naples Federico II. He is in the Dependable Systems and Software Engineering Research Team since 2007. His research interests focus on software testing and on dependability of software systems. He co-founded Critiware (www.critiware.com) a company working in critical systems engineering since 2011. He is involved in several projects and currently coordinates an MSCA RISE European project (μDevOps). He is senior member of IEEE.

Francesco Palmieri is a Full Professor at the University of Salerno. His research interests include advanced networking protocols and security. He has been the director of the Networking Division of the University of Naples "Federico II" and contributed to the development of the Internet in Italy as a senior member of the TSA Committee and of the CSIRT of the Italian NREN GARR. He serves as the editor-in-chief and editorial board of several international journals.