The Journal of Supercomputing
https://doi.org/10.1007/s11227-023-05098-2

MECInOT: a multi-access edge computing and industrial internet of things emulator for the modelling and study of cybersecurity threats

Sergio Ruiz-Villafranca[1] · Javier Carrillo-Mondéjar[2] · Juan Manuel Castelo Gómez[1] · José Roldán-Gómez[3]

Accepted: 3 February 2023

Abstract

In recent years, the Industrial Internet of Things (IIoT) has grown rapidly, a fact that has led to an increase in the number of cyberattacks that target this environment and the technologies that it brings together. Unfortunately, when it comes to using tools for stopping such attacks, it can be noticed that there are inherent weaknesses in this paradigm, such as limitations in computational capacity, memory and network bandwidth. Under these circumstances, the solutions used until now in conventional scenarios cannot be directly adopted by the IIoT, and so it is necessary to develop and design new ones that can effectively tackle this problem. Furthermore, these new solutions must be tested in order to verify their performance and viability, which requires testing architectures that are compatible with newly introduced IIoT topologies. With the aim of addressing these issues, this work proposes MECInOT, which is an architecture based on openLEON and capable of generating test scenarios for the IIoT environment. The performance of this architecture is validated by creating an intelligent threat detector based on tree-based algorithms, such as decision tree, random forest and other machine learning techniques. Which allows us to generate an intelligent and to demonstrate, we could generate an intelligent threat detector and demonstrate the suitability of our architecture for testing solutions in IIoT environments. In addition, by using MECInOT, we compare the performance of the different machine learning algorithms in an IIoT network. Firstly, we present the benefits of our proposal, and secondly, we describe the emulation of an IIoT environment while ensuring the repeatability of the experiments.

Keywords Industrial internet of things · Multi-access edge computing · Machine learning · Cybersecurity

✉ Sergio Ruiz-Villafranca
sergio.rvillafranca@uclm.es

Extended author information available on the last page of the article

Published online: 03 March 2023

1 Introduction

The industrial environment has undergone massive changes since the First Industrial Revolution, with the latest revolution being called Industry 4.0 or the Industrial Internet of Things (IIoT). Emerging technologies and implementations such as the Internet of Things (IoT), Artificial Intelligence (AI) and 5 G networks have converged with traditional Operational Technology (OT) protocols and devices to improve the performance and efficiency of the enterprise [1]. Some recent proposals from the research community [2] have even started mentioning Industry 5.0, which has the same basic goal of solving the problems related to the integration of IIoT into traditional industry, but while the approach followed in Industry 4.0 has been to implement an independent integration for each company, focusing on the advantages and the functionalities of the technologies related to IIoT, Industry 5.0 adopts a different focus, trying to highlight the importance of innovation and research to support the industry in its service to humanity. Thus, Industry 5.0 pays particular attention the points related to resilience, sustainability and human-centric values, with the support of the technologies considered in Industry 4.0 adding the advances in biotechnology and renewable and energy-efficiency technologies [3].

One of the proposed solutions is Multi-Access Edge Computing (MEC), which has been created as the successor to Edge Computing and is designed to bring the advantages of cloud computing closer to companies, and to give support to emerging technologies. Its main objective is to improve the network performance of cloud applications, as well as to enable the implementation of new applications that are delay-sensitive, such as autonomous driving or virtual reality applications. In addition, MEC facilitates the implementation of IIoT since the concept itself already brings together some of the technologies considered in it [4]. However, the convergence of OT protocols and Information Technology (IT) protocols, together with the inclusion of IoT applications and web applications, can produce cybersecurity risks if this process does not follow the good practices required in implementation. Moreover, the risks are higher in critical infrastructures, which could increase losses for companies and states and have a debilitating effect on sectors such as security, national economic security, national public health or safety. Another aspect to take into account is that many devices found in these infrastructures are not likely to be updated periodically [5]. As a result, these infrastructures are a perfect target for attackers and their strategies, as their techniques change and evolve rapidly. This insecure context can be found in others environments, such as industrial plants, private health care, or industries related to smart city functionalities, each of which has specific requirements that are different from those of critical infrastructures [6, 7]. Thanks to the use of Machine Learning (ML) and AI algorithms, cybersecurity researchers can use the data left by attackers or researchers themselves on their network or application to train multiple tools. As a way to monitor and avoid (or reduce) the impact of these attacks, these tools are becoming the perfect solution for the cyberdefence department of companies. Recently, Federated Learning and Adversarial

Networks have been improved in order to use the data collected by the devices in the network, and then train the models with them, which results in these models being more sensitive to changes and new attacks [8]. Unfortunately, evaluating the feasibility and performance of these threat detectors when deployed in low-resource environments such as IIoT is not straightforward, since an architecture is needed for generating reproducible experimentation scenarios while taking into account the specific characteristics of IIoT environments.

There are some related works that have tried to address this problem. The first one is openLEON [9], which is used as the basis of our proposal. However, openLEON does not implement any service related to IIoT. Another interesting proposal is the IIoT Testbed [10], which is an emulator focused on the deployment of an IIoT application to monitor different metrics. The limitation of this tool is the low level of flexibility in its scenario, as it is focused only on the IIoT application deployed, thus making it impossible for use in the study and modelling of IIoT networks and the related experimentation, and it requires a tool to design and deploy networks that follow the IIoT paradigm.

With these issues in mind, this paper presents MECInOT, an emulator built on top of OpenLEON [9] for facilitating the modelling and deployment of an IIoT topology. MECInOT allows the deployment of OT and IT applications up to the edge data centre topology. MECInOT has also been designed with special attention to cybersecurity research, providing different tools for the proposed scenarios in which different attacks on the network and application layers can be carried out. In this regard, we use MECInOT to collect network data while attacks are taking place in the emulation of the scenarios, and then to implement an Intrusion Detection System (IDS) based on different ML algorithms. The use of ML in threat detection offers an advantage over the use of rules found in most commercial IDS. These advantages are due primarily to the inability of rule-based systems to detect unknown or new attacks, in addition to the difficulties that such systems have in operating in highly dynamic environments such as IIoT scenarios [11]. Therefore, the trend is towards the machine-learning-based approach, and in this work we intend to implement the IDS to validate our emulator in the field of cybersecurity research by adopting such an approach.

MECInOT, which is the emulator present in this work is available in a Github[1] repository, and the dataset used is also available online[2].

The rest of this paper is organised as follows. In Sect. 2, we review the proposals from the research community regarding emulators for MEC, the IoT and the IIoT. Section 3 describes the technical background of our study. The emulator developed in this experiment is presented in Sect. 4, together with the applications for each OT, IoT and IT protocol and the attacks implemented on it. In Sect. 5, we evaluate the performance of the proposal, using the CPU usage parameter as a reference, and we present the attacks implemented on the emulator and the generation of malicious

[1] Link to Github repository: https://github.com/C4denaX/MECInOT.

[2] Link to dataset: https://data.mendeley.com/datasets/xstyjwrc5r/draft?a=440538fd-e139-4e06-8885-107785f51807.

data. By using these data, in Sect. 6 we describe the methodology followed to make an IDS using ML algorithms and analyse the results obtained by it. Finally, Sects. 7 and 8 correspond to the conclusions that can be drawn from this experiment and the future work planned, respectively.

2 Related work

This section presents a study of the most important pieces of research focusing on the emulation of IIoT and MEC scenarios.

A work which implements an IIoT context in its design is the IIoT Testbed emulator [10]. This emulator allows the emulation of IIoT applications based on Data Distribution Service (DDS) middleware, allowing the users to manage different industrial processes, including the modification of multiple parameters, such as Quality of Service (QoS), of the different services and functionalities emulated. The creation of new IIoT processes and their management is performed by using a web interface, making the use of the emulator easier for the users without the need to understand the whole architecture of the emulator. However, the IIoT Testbed is a very limited tool in terms of its functionality and flexibility when it comes to creating new conditions for the scenario that it proposes. This is mainly because the tool is focused on the management of the process and its related data. In fact, the IIoT Testbed emulator cannot success fully meet the needs of different experiments in creating multiple scenarios and IIoT topologies with different devices and protocols used by the applications. Also, MEC is not considered during the design and implementation of this emulator. This is the case of rest of the proposals described in this section, but in the opposite sense, that is to say that Edge Computing is considered but industrial protocol applications are not.

A work that considers MEC in its proposal is [12], which establishes a development environment for the deployment of applications based on Fog Computing and MEC architectures. This emulator allows the creation, design and definition of networks, the deployment of multiple edge nodes, and the implementation of IoT applications. Moreover, the emulator allows the analysis of the performance of each node deployed and of the network that interconnects them, in order to check whether there are any errors or misconfigurations in the scenario defined or the performance of the application deployed. However, to execute Fogify it is necessary for the user to have an in-depth knowledge of how to use it, as well as some fundamentals on the tool in charge of its deployment, namely Docker Swarm. In addition, this type of tool can lose the support of the community in favour of other popular ones such as Kubernetes. Furthermore, the emulator does not support the inclusion of OT protocols or new IT or IoT protocols, making it impossible to create a heterogeneous IIoT scenario.

Using the emulators Containernet and Maxinet, which are extensions of the Mininet emulator, [13] presents a solution for the creation, design, definition and deployment of virtual networks that users consider. These emulators use the Python programming language, which helps the users with the deployment of the networks through the creation of a single script. Containernet deploys the final devices as

containers, providing the flexibility of this technology when creating for example, applications, device or server functionalities. The main problem of Fogbed is that the emulator does not include support for mobile networks, which is one of the principal characteristics of the MEC paradigm. Also, the emulator is not IIoT-focused, making it impossible to implement an MEC-IIoT scenario using just this emulator. In addition, Fogbed uses the default version of Containernet, which does not allow redundancy in the MEC topology, with all that this implies, such as the errors that could appear with disconnections between edge nodes when the complexity of the topology and the number of applications and services increase.

The last emulator found in the literature that can allow the deployment of these kinds of networks is openLEON [9]. This emulator solves the problems found on Fogbed emulator, because openLEON implements a network controller that allows the use of spanning-tree in the topology, thus making it possible to implement redundancy between network devices and edge nodes. In addition, the emulator enables the use of srsLTE, which together with the corresponding hardware allows connectivity with Long-Term Evolution (LTE) networks in MEC topology. Therefore, openLEON is considered one of the most appropriate emulators for use in the deployment of MEC-IIoT scenarios. However, as was mentioned with Fogbed, openLEON is only focused on the deployment of the MEC topology, and IIoT applications and devices are not considered. This means that experimentation, research and security checking in the context of IIoT are impossible for users. In order to address this shortcoming, our MECInOT proposal uses openLEON as a basis for MEC topology deployment, while integrating the deployment of IIoT networks with their corresponding services, applications and devices. This is an added functionality that to the best of our knowledge, has not been included in any proposals in the literature.

As far as we know, there are no other works that address the emulation of MEC-IIoT environments. This fact indicates the important contribution that our work can make to the state of the art. Also, MECInOT introduces new functionalities and characteristics such as MEC integration, flexibility to introduce new IIoT protocols and IIoT services, and the integration of real devices and physical resources in the topologies.

As a summary, Table 1, shows the characteristics of the different pieces of research reviewed in this section.

3 Background

In this section some technical concepts related to the emulator and machine learning algorithms are described.

3.1 Multi-access edge computing

MEC, a concept standardised by the European Telecommunications Standards Institute (ETSI), is considered the evolution of Edge Computing and is designed

Table 1 Summary of the proposals from the research community

Proposal	Technologies used	Edge oriented	IIoT oriented	Support for mobile networks	Flexibility to add new functionality	Year
IIoT testbed [10]	Django Framework (Python)	No	Yes	No	It is not possible	2018
Fogify [12]	Docker Swarm	Yes	No	No	It is possible to define new IoT applications	2020
Fogbed [13]	Containernet	Yes	No	No	Edge servers can run new services	2018
OpenLEON [9]	Containernet (modified), srsLTE	Yes	No	Yes	Edge servers can run new services	2019
MECInOT	OpenLEON, Docker-Compose	Yes	Yes	Yes	It is possible to add new protocol services and applications to Edge Servers and to IIoT topologies	2022

to improve the performance of communications in the Cloud Computing environment and IoT and IIoT technologies. One of the most important characteristics of MEC is the use of virtualisation technology to provide computational resources to applications and the heterogeneous network management that MEC must support [4]. As it brings computational functions closer to end users than Cloud Computing, it makes it possible to reduce the latency in communications and avoid network saturation in the Cloud Computing providers. This means that users perceive a better performance in the applications, and it allows the development of new applications that previously were impossible with such limitations. Also, with the use of the new 5 G mobile networks and the future 6 G ones [14], a reduction in latency between devices can be achieved, making it possible to deploy real-time applications on this kind of architecture.

To make it possible to implement MEC, it is necessary to use multiple virtualisation technologies, namely:

- **Network Function Virtualization (NFV).** The implementation of IIoT in traditional factories means that a large number of IoT devices and applications are connected to industrial environments. Many of these devices use different communication protocols, which in some cases will be proprietary, because these devices are designed for specific contexts and scenarios, which in turn means using specific network devices to manage the proprietary protocol traffic. NFV allows network admin to avoid the use of these specific network devices to manage the heterogeneous and proprietary traffic, reducing the cost associated with the deployment of the different network devices and their number, thanks to using generic network devices. Basically, NFV is software that can be installed on any device that can receive and send traffic. ETSI defines and determinates the applications that NFV can support, such as connectivity functionalities, Dynamic Host Configuration Protocol (DHCP) or Network Address Translation (NAT) [15].
- **Software-Defined Networking (SDN).** Traditionally, in communication networks the data layer is defined as the management part of the network of the traffic generated by the users, while the control layer manages the routing processes. Both layers work together on the network devices, which produces a slow-down in high-demand networks. The growth in complexity of the networks has shown that this solution is neither scalable nor flexible. To partially solve this issue, network administrators manually reconfigure network devices by using scripts, which leads to misconfigurations, but in any case this becomes an immeasurably difficult task in complex networks [16].

  In view of these misconfigurations from network administrators, SDN tries to solve this problem by disassociating the two layers. Thus, a specific network architecture is defined, and the allocation of the network resources is established thanks to network virtualisation [17].
- **Network slicing.** This technique defines different virtual networks depending on a criterion, and the allocation of the network resources is determined according to the needs established for each virtual network. Network Slicing allows the generation of multiple custom networks using the network

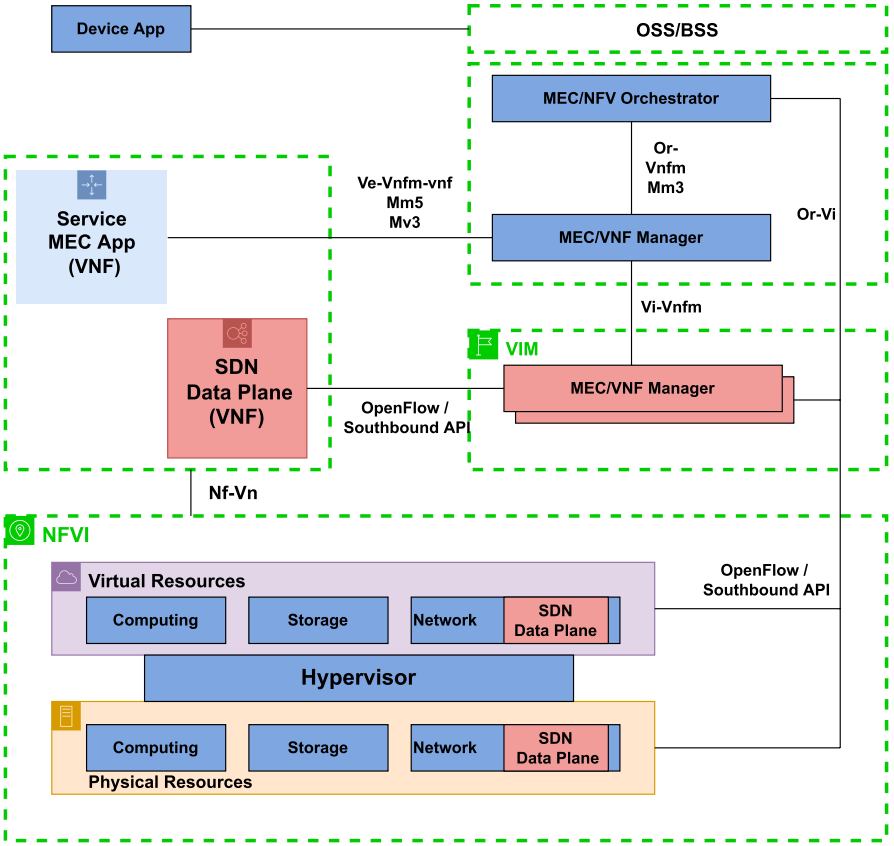

Fig. 1 Reference MEC architecture [4]

resources deployed physically. This provides flexibility when allocating the network resources according to the needs of the virtual network, or even if then occurs an unexpected demand for resources at a particular time [4].

- **Service Function Chaining (SFC).** The transition from traditional networks to software-defined networks and virtualisation is a huge effort for IT teams [18]. Thus, the principal goal of SFC is to facilitate this transition dynamically. The concept of SFC is quite similar to NFV, the principal difference is that SFC solves the problem that can appear when a service is provided individually, and it belongs to a service chain. Thus, SFC manages the services that are running on the network devices and which can be used at a specific moment [17].

Figure 1 shows an example of implementation of a MEC architecture using the basic NFV and SDN services explained above, illustrating how communication between the SDN controller and the NFV controller is performed in this architecture.

3.2 Machine learning algorithms

Machine learning has grown rapidly in recent years in the context of computing and whole data analysis performed via other technologies such as the IoT. ML uses this new information generated by analysis to develop applications with new functionalities in an intelligent manner. Thus, ML provides the system with the processes and tools to automatically extract knowledge from the data that it receives without the need to be programmed [19].

As mentioned above, we use a machine learning approach because several papers consider it to be best in environments where there are unknown threats, especially if these environments are highly changeable and heterogeneous, due to the difficulty of manually establishing rules for each type of attack. There are comparisons that demonstrate the superiority of machine-learning-based approaches in these contexts [20, 21]. In addition to the better performance of a machine-learning-based IDS, it also offers certain advantages, such as:

- Accuracy: Machine learning algorithms can analyse large amounts of data and identify patterns and anomalies that may be difficult for humans to spot or define explicitly.
- Flexibility: Machine learning-based IDS can continuously learn and adapt to new types of attacks and changing network environments.
- Anomaly detection: By modelling the normal behaviour of the system, it is possible to detect anomalies that are completely impossible to detect for rule-based systems.

Supervised ML techniques are considered to be those algorithms which provide intelligent models with the construction of general hypotheses and patterns [22], using external data instances to address the prediction of future instances from a similar data input. The principal function of these techniques is to develop a prediction model for solving classification and categorisation problems, establishing the possible category of the incoming data [23]. This can be solved with binary classification in the case of there being only two categories to predict, or multiclass classification, which normally refers to the prediction of more than two categories. One must also consider the special case in which the incoming data can be categorised into multiple categories at the same time [24].

Some algorithms that give good results in multiclass classification problems with tabular data are the following:

- **Decision Tree (DT).** DT is a well-known supervised ML algorithm [25], and it is also used for regression problems. This algorithm works by using tree structures, starting from a root node and sorting down to certain leaf nodes, whose number depends on the data and the categories. DT performs the classification with the instances, which are classified by checking the attribute established and defined at each node of the tree until reaching a leaf node which shows the category of the entry data. The splitting is carried out by

using two different criteria, namely gini and entropy, whose mathematical equations are given by Eq. 1 and Eq. 2 [26].

$$Entropy : H(x) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i) \tag{1}$$

$$Gini(E) = 1 - \sum_{i=1}^{c} p_i^2 \tag{2}$$

- **Random Forest (RF).** The RF classifier is an ensemble classification model used in various areas of application [27]. This algorithm makes a parallel ensemble fit many DTs in parallel using different datasets or subsamples of the same dataset to train them. The output is the majority solution of the trees or the average result [26].
- **Naive Bayes (NB).** The NB algorithm is based on Bayes theorem, which makes the assumption of the independence between each pair of features that are allocated to the entry data of the model, and its definition is given by Eq. 3 [28]. The principal advantage compared with other approaches is that NB only needs a small amount of training data to quickly estimate the parameters needed. However, the main problem of this algorithm is that its performance can be affected by the strong assumptions of feature independence [26].

$$P(A \| B) = \frac{P(B \| A) \cdot P(A)}{P(B)} \tag{3}$$

- **Stochastic Gradient Descent (SGD)** [29]**.** SGD is an iterative algorithm for the optimisation of the results with an objective function with appropriate properties. This algorithm allows a reduction in the computational cost of training, especially in high-dimensional optimisation problems, allowing faster iterations at the cost of a lower convergence rate. SGD is usually applied to problems of text classification and natural language processing. However, the algorithm can produce worse results for feature scaling and needs the hyperparameter tuning of some parameters, such as the number of iterations and the regularisation parameter.
- **Support Vector Machine (SVM)** [30]**.** SVM is a supervised ML algorithm used in various fields and kinds of problems, especially classification problems. SVM uses statistical learning approaches and classifies the input data by determining a set of support vectors. The main goal of SVM is find the optimal hyperplane for the classification of new data. In our work, the SVM model developed is optimised with SGD.

There are other possible state-of-the-art algorithms, such as neural networks and their different architectures [31]. However, in classification problems with tabular data they present certain disadvantages, especially in IoT and IIoT environments, where there are resource constraints. These limitations are as follows:

- Complexity: Neural networks can be complex models to train and require considerable computational resources. This can make them difficult to implement and run, especially for large datasets.
- Overfitting: Neural networks are prone to overfitting, especially when working with small datasets. This means that they may perform well on the training data but may not generalise well to new data.
- Lack of interpretability: Neural networks can be difficult to interpret and it can be hard to understand how they make predictions. This can make it difficult to understand the factors that are driving the predictions and to identify any potential biases in the model.
- Time-consuming: Training a neural network can be time-consuming, especially for large datasets. This can make it difficult to use neural networks in real-time applications, where quick predictions are necessary.
- Require more data: Neural networks generally require more data to achieve good performance than other machine learning algorithms. This can be a disadvantage if the dataset is small or if there are certain types of data that are difficult to collect.

Overall, while neural networks can be powerful tools for tabular data classification, they can also be complex and time-consuming to work with and may not be the best choice in all cases. Other machine learning algorithms, such as decision trees or random forests, may be more suitable in certain situations [32].

For these reasons it has been decided not to validate the emulator with neural networks, although future experimentation in which the architecture is validated with deep learning architectures of low computational cost would certainly be of interest [33].

4 MECInOT proposal

This section provides the details of the design and the methodology that have been followed in developing MECInOT.

4.1 Design

When designing MECInOT, we considered that the emulator should include the following features:

- **Offer a realistic emulation of physical scenarios.** It should allow to users to carry out experiments and proofs of concepts, and to obtain data in the same way as when using a real topology, without the need to use physical devices.
- **Provide the possibility and flexibility to develop different scenarios depending on the research needs.** Thanks to the use of the virtualisation of devices using containers, it is possible to easily modify the number of devices, their

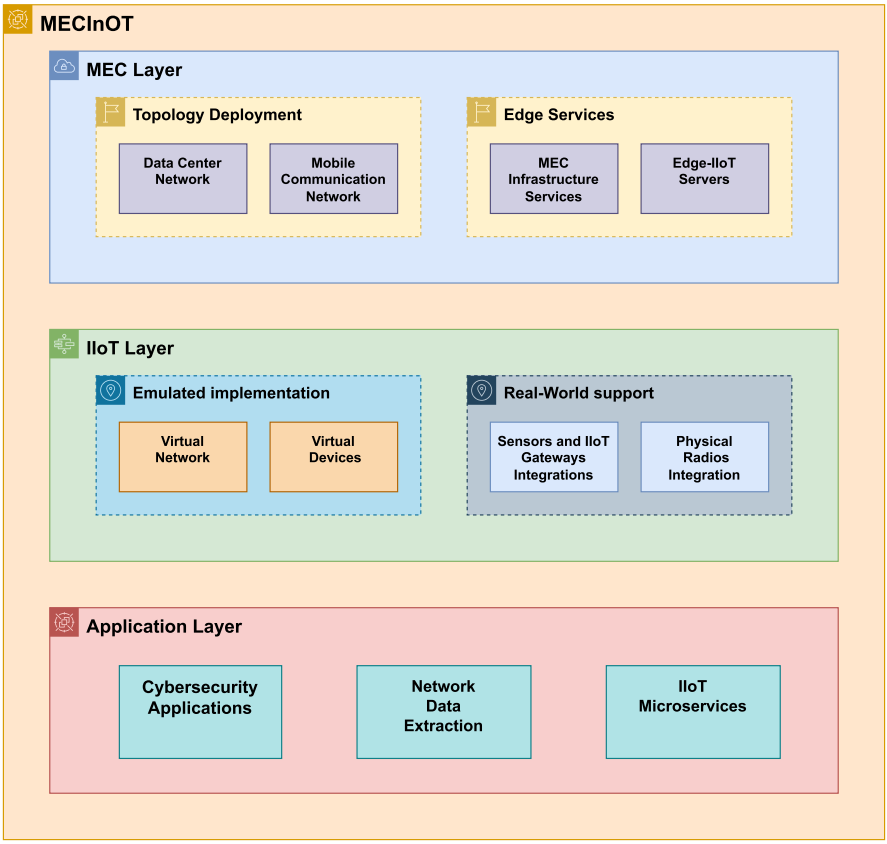

Fig. 2 MECInOT architecture

functionality, or even the definition and deployment of multiple virtual subnetworks, with each one having a specific function or implementation.

- **Facilitate the insertion of real devices in emulated scenarios.** The possibility may arise in an experiment of having to use real devices in an emulated scenario in order to validate the performance results. In this proposal it is possible to introduce new metrics such as collisions, packet delays or device performance. Therefore, thanks to the virtualisation technologies used by MECInOT, real devices can be integrated into the scenarios deployed by the emulator, new network devices such as IoT gateways can be implemented. Also the integration of physical radio for specific protocols such as Zigbee, Z-Wave, or 6lowpan is available.
- **Enable cybersecurity research.** The main goal during the design of MECInOT was to develop an emulator that make it possible to carry out cybersecurity research in the MEC-IIoT context. Consequently, the emulator provides the users with different scripts and a malicious container that allow them to easily perform different types of attacks in their scenarios, generate malicious data, and

check the impact that these attacks have on the topologies. Also, through the design of the emulator, these attacks can have a consequences on the real devices connected to the emulated scenarios, providing more options for validating and obtaining results.

Figure 2 shows the MECInOT architecture, in which we can see the different functionalities considered during the design of the emulator.

4.2 MECInOT deployment methodology

In a real Industry 4.0 scenario, it is mandatory to deploy each device of the industrial network or enterprise network, and provide them with an individual network configuration that covers the specific needs of the scenario. In MECInOT, the first step is to virtualise the machine which creates, manages and deploys the industrial private network, as well as the machine which deploys the MEC architecture. Next, the communication between the industrial network and the MEC architecture is defined. To make this possible, both the machines must have the IP address of the same private subnetwork, configuring the network adapter of the virtual machines in bridge mode.

It is necessary to define the subnetwork addresses that are going to be used by the virtualised IIoT devices and the MEC virtual subnetwork. In MECInOT, the emulated IIoT devices are deployed using the container technology tool Docker-Compose, which facilitates and speeds up the deployment and the implementation of changes in the virtual scenarios. For this reason, the configuration of the virtual IIoT networks is given by this tool, which allows the users define multiple subnetworks for each scenario. For MEC architecture deployment, the network configuration used is the default one established by the openLEON emulator with multiple Edge servers that will run the multiple IIoT services of the scenarios. In addition, it is necessary to establish a correct routing configuration on the IIoT network machine in order to enable correct communication between IIoT subnetworks and the MEC subnetwork. Once this process has been carried out, the communication between IIoT and MEC hosts should be tested to check whether it is correct.

The last step consists in deploying the MEC-IIoT architecture in which the experiments are going to be performed. This architecture is comprised of the IIoT devices, which are already implemented in the emulator scenarios provided, and the network topologies described in the following sections.

Figure 3 shows the communication the between different steps mentioned above.

4.3 IIoT topology

With the purpose of emulating the network of an Industry 4.0 factory, MECInOT implements a business network using the interface created by Docker-Compose to deploy the different Docker containers, each of which corresponds to an IIoT device in the virtual factory. In addition, the emulator allows the users to define and

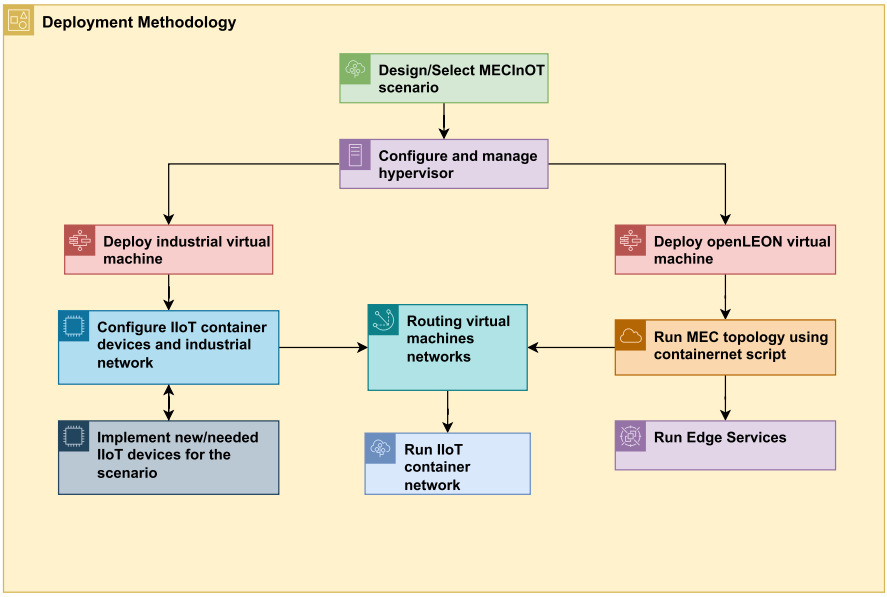

Fig. 3 MECInOT deployment methodology

implement different subnets with the most common protocols that can be found in an IIoT scenario, with these being:

- **IoT protocols.** MECInOT includes IoT applications that rely on the most commonly used protocols in this context: Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP) and Advanced Message Queuing Protocol (AMQP) [34, 35].
- **OT protocols.** The protocols added to MECInOT are versions of the industrial protocols which use the TCP transport protocol: Modbus/TCP, which is the most widely used industrial standard and protocol [36]; S7COMM, which is the proprietary communication protocol used by the Programmable Logic Controller (PLC) of Siemens [37]; and Open Platform Communications United Architecture (OPC UA), which is considered as one of the protocols that allows the convergence of OT and IT protocols [35].
- **IT protocols.** This group of protocols includes those that are widely used by Internet users, such as the Hypertext Transfer Protocol (HTTP) [38].

As mentioned above, the design of MECInOT is focused on enabling cybersecurity research in MEC-IIoT scenarios. Consequently, the emulator provides a set of tools that allow users to carry out attacks in these scenarios, thus making it possible to evaluate the impact of such attacks, as well as the risks and the costs that these could entail in the particular context studied. This toolset is currently built into an attacking node using the Kali Linux distribution image which ca be run in a Docker Container on the IIoT network.

To evaluate the impact of a possible attack on the infrastructure deployed with the emulator, different types of attacks are employed. These attacks belong to the following categories:

- **Packet manipulation.** To carry out this type of attack, the attacker node must use the Man in the Middle attack to capture the packets and modify them. The changes made to the packets are focused on the data field of each protocol in the emulated scenario. The rest of the fields do not change.
- **Brute force.** The attack uses a dictionary of users and passwords to make multiple login attempts to try and access devices and services. Thus, MECInOT has a script that performs the dictionary brute force attack on a login form of a web server that can be running on a PLC or edge node.
- **Attacks with payloads in HTTP frames.** These attacks aim to exploit the Shellshock vulnerability [39], which is a software bug found on some web servers that can be used to gain access to the machine on which the service is running. A script is made to automatise the process of gaining access to the machine shell. This vulnerability is exploited by sending an HTTP packet with a modified User-Agent field that contains the specific payload.
- **Network scanning.** The emulator includes a tool that allows different scanning methods to be carried out automatically. The scanning methods implemented are TCP SYN, TCP connect, UDP, TCP NULL, TCP FIN, TCP XMAS, and TCP ACK.
- **Denial of Service (DoS).** For this type of attacks, traditional denial of service methods have been implemented, such as Ping of the Dead [40], and methods based on the flood or saturation of the port with a massive sending of TCP packets. In addition, these methods have been adapted depending on which protocol is targeted. In particular, for the AMQP protocol, fake devices are included in order to overflow the message queue, resulting in legitimate user not being able to establish communication with the queue. With regard to CoAP protocol, this uses UDP as a transport protocol, which allows the implementation of the UDP amplification attack [41]. This attack consists in using a simple modified request to a method on a CoAP server to be received by the target machine. As the attacker sends lightweight packets, this makes it possible to cause a denial of service for some of the devices in the network using the CoAP server, which sends larger packets than the attacker.
- **Malicious device injection.** In this attack, the attacker tries to find a default configuration or a bad implementation of an MQTT broker in order to obtain more information from it. This process is automatised by using a script which emulates a new device that tries the connection to the specific topic on the MQTT broker as a way to read whole messages that pass through it.

4.4 MEC topology

As was mentioned above, the MEC topology is deployed using the openLEON emulator. In this emulator there two key components:

- **Data centre.** openLEON implements this part of the topology by using the Containernet emulator [42], which is an extension of the Mininet emulator [43], thus allowing the creation of topologies whose hosts are implemented with containers. Secondly, the architecture of the topology has a 3-level hierarchical network structure, which is typical of a 3-tier data centre. The topology has two core switches, and two aggregation switches for each one. Also, the architecture has 64 hosts connected between the switches found on the Top of Rack level. This topology structure provides redundancy between network devices and hosts, thus trying to avoid communications errors in the network. In order to do this, it is useful to implement a spanning-tree protocol [44] on the network devices. Since the default SDN controller implemented in Mininet does not support this functionality, openLEON developers decided to implement and use RYU [45], which is a module implemented with Python that provides support for OpenFlow to be integrated in the topology.
- **Mobile communication.** As was mentioned in Sect. 2, this is a crucial MEC component. The protocol used to provide mobile communication is LTE, and this is achieved by using the srsLTE emulator [46]. In order for this emulator to run correctly it needs a set of hardware, such as antennas and mobile LTE stations. With this module it is possible to establish communication between the mobile devices connected via LTE with the MEC-IIoT topology.

4.5 MECInOT distributed deployment

With the aim of taking advantage of the container and Mininet virtualisation technologies, a distributed deployment is also defined in order to be able to use MECInOT on a High Performance Computer (HPC) or cluster, and thus obtain a more realistic and better performance than when using a single computer. This kind of deployment allows the use of multiple MEC and IIoT topologies on the distributed nodes of a cluster. For this implementation, it is recommended to use a container orchestrator for the management of the containerised IIoT topologies. In addition, it is possible to deploy multiple MEC topologies without the need to use the virtual machine by only employing the Containernet implementation of openLEON.

Figure 4 presents a logical implementation of this type of deployment, also showing the intercommunication between the nodes and topologies providing a correct functionality and connection between the different parts of the emulator.

5 MECInOT evaluation

In this section some examples of scenarios that can be designed and deployed with MECInOT are described. In addition, we present an evaluation based on the cost of CPU usage metric for the physical machine on which the emulator is run. Finally, a cybersecurity analysis and proof of concept are carried out to check whether the emulator can contribute to this research field with the MEC-IIot scenario deployed.

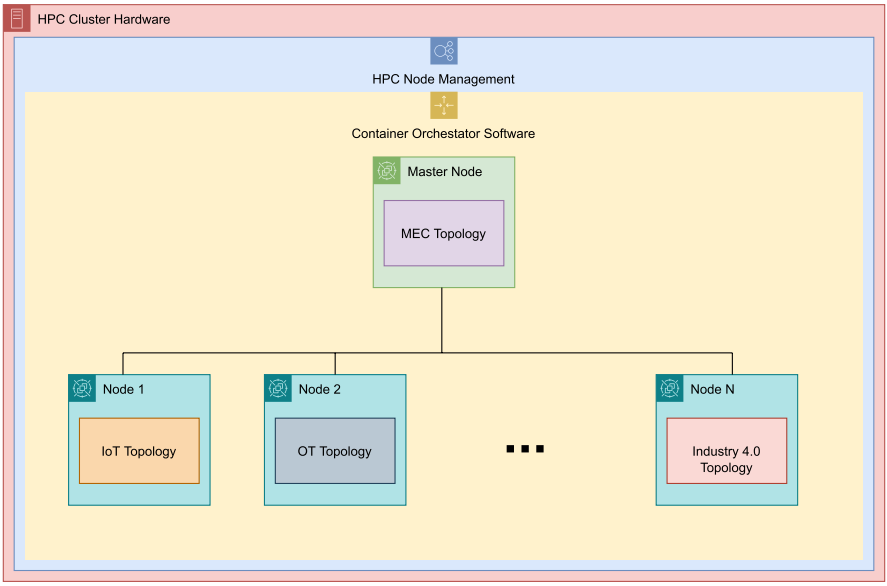

Fig. 4 MECInOT logical distributed architecture

5.1 Hardware setup

For the evaluation of the proposal described in Sect. 4 we used a laptop with an Intel i7-10875 H 2.30GHz CPU with 32 GB RAM memory, running Windows 10 20H2. In order to execute the virtual machine, the hypervisor VirtualBox 6.1.16r was used. Finally, a Raspberry Pi Zero 2 was included as additional hardware in order to add the possibility of deploying an IoT gateway in our scenarios.

5.2 Industrial OT scenario

In order to evaluate this scenario we only deployed devices that specifically use industrial or OT protocols, as mentioned in Sect. 4.3.

The scenario studied for the **OPC UA protocol** consisted of a client node which was in the industrial topology, and a server node that was allocated in the MEC topology. The basic functionality between the two nodes was a communication based on the reading of a random string that was generated by the server node at a random time between one and nine seconds.

For the case of the **S7 protocol**, two client nodes were deployed, one for reading and the other for writing, and these nodes were allocated in the industrial network. A server node was also deployed in the scenario, but it belonged to the MEC topology. The functionality provided by these nodes was very similar

to that mentioned for the OPC UA protocol. The writing client wrote a random string on the server, and then this was read by the reading node at a random time.

For the ModBus/TCP protocol, the deployment involved two clients allocated in the industrial topology, one for writing and the other for reading, and a server node that was deployed in the MEC network. In this scenario, the writing node wrote a sequence of values, which could be alternately True or False, to the server, with the reading node periodically reading the values stored on the server.

In order to provide the emulator with cybersecurity testing capabilities, an attacker node is deployed in the industrial topology, and it can communicate with the rest of the industrial devices and MEC servers. This node, as well as the tools, scripts and attacks implemented, is described in Sect. 4.3.

5.3 Industry 4.0/IIoT scenario

This scenario is an extension of the scenario detailed in Sect. 5.2, and introduces communication between devices using IoT protocols and the emulation of users that operate with IT protocols. This allows researchers and emulator users to implement and deploy an example of an Industry 4.0 factory, which is connected to different network services. In this scenario, the IoT traffic passes through an IoT gateway which is deployed in the business network. In addition, an attacker node is introduced in order to provide the user with the option of performing cyberattacks with the new devices included.

The MQTT protocol is implemented with the aim of emulating a factory that has multiple sensors sending the temperature in degrees Celsius every three seconds. The IoT gateway is given the role of MQTT broker, acting as an intermediary between the communication with the MQTT subscriber, which belongs to one of the edge nodes in the MEC network, and the rest of the devices.

In the case of the CoAP protocol, a client is deployed in the industrial network and it sends multiple messages to the IoT gateway, which performs the function of master/server node, with these messages being read by another CoAP client allocated in MEC topology.

For the AMQP protocol, a client node is deployed in the industrial network and it publishes a random number between 33 and 126 in the RabbiMQP queue that is running on the IoT gateway. These messages are received by an AMQP client allocated in the MEC topology.

The IT protocols that are considered and included in the scenario are the HTTP and Hypertext Transfer Protocol Secure (HTTPS) protocols. The former implements a login form for the factory users, and the latter does so with a video streaming server. On the user side, they are specifically in the industrial network and have been implemented using Python scripts that emulate their normal behaviour using the IT applications mentioned above.

In Fig. 5, a logical schema of communication between the different parts in this scenario is shown. In addition, it describes how communication is performed depending on the type of traffic.

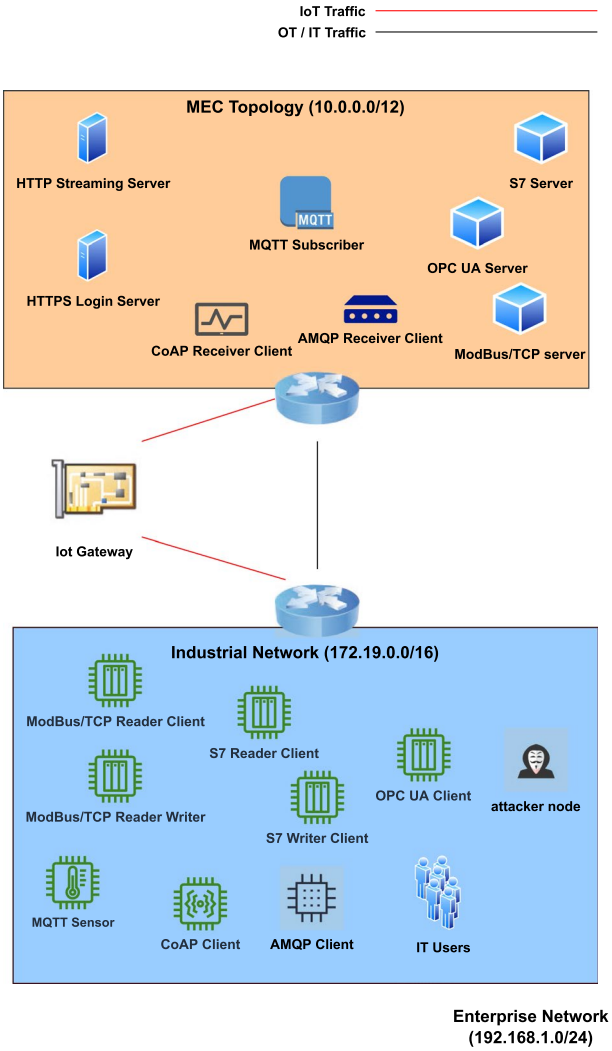

Fig. 5 Logical network schema of Industry 4.0/IIoT scenario

5.4 Performance analysis

Now we have described some of the scenarios that can be created and deployed with MECInOT, we analyse its performance in these scenarios. This analysis considers the cost in terms of computational resources associated with the deployment of the Industry 4.0 scenario on the machine, since it is the scenario with the highest number of devices and services running simultaneously. The metric used is the CPU usage of the host machine running Windows 10, and the tool used to obtain this metric is System Monitor, which is a piece of native software for obtaining information regarding hardware conditions and process metrics.

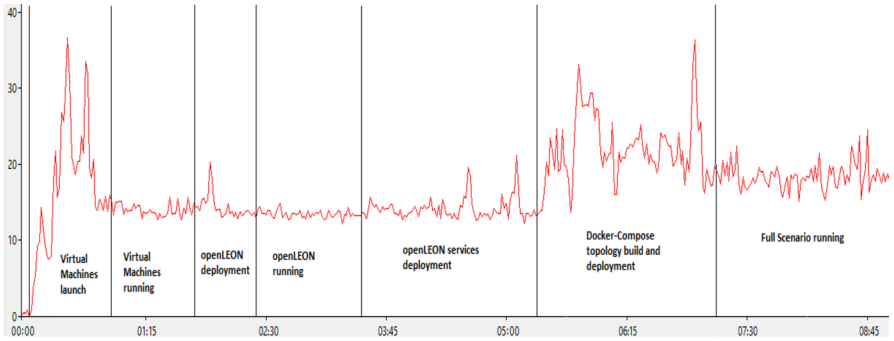

Fig. 6 Evolution of CPU usage with the deployment of the Industry 4.0 scenario. The X-axis represents the running time of the experiment in minutes:seconds format, and the Y-axis represents the CPU usage percentage

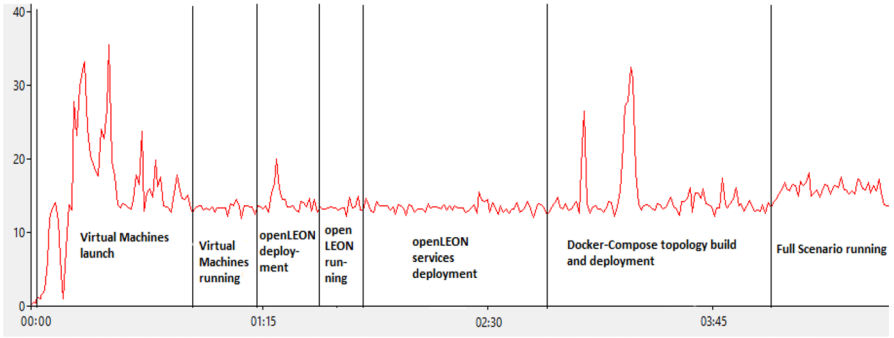

Fig. 7 Evolution of CPU usage with the deployment of the OT scenario. The X-axis represents the running time of the experiment in minutes:seconds format, and the Y-axis represents the CPU usage percentage

In Fig. 6, the evolution of CPU usage over nine minutes (the execution time of the scenario) is shown. The metric is measured from the launch of the virtual machines to the stopping of the emulator. We can observe the different load rate changes during the deployment of the different parts of the emulator, and the fact that the maximum CPU usage occurs in the building and deployment of the Docker-Compose containers.

In order to check the flexibility and how load usage varies depending on the complexity of the scenario deployed, a comparison is made with the scenario that only deploys the applications that uses OT protocols and the above mentioned one. In this case, the OT scenario is simpler than Industry 4.0 and allows us to make the comparison desired. Figure 7 shows the evolution of the CPU load usage during five minutes of the execution of the OT scenario.

By comparing Figs. 6 and 7, we can see that the resources needed to run a given scenario vary in accordance with its complexity. However, by measuring

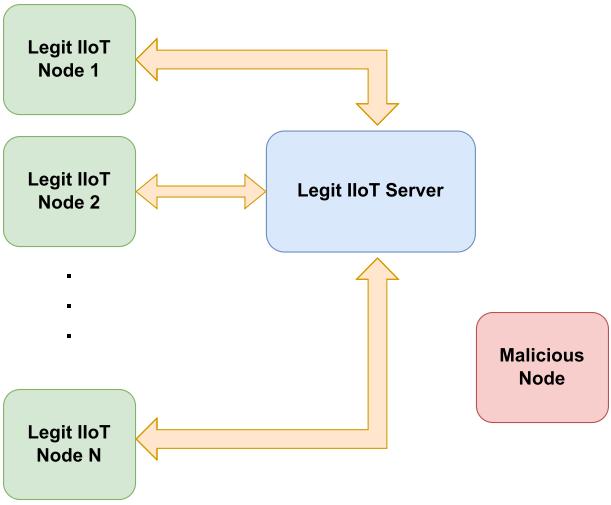

Fig. 8 IIoT baseline schema for its cybersecurity analysis

the median CPU usage in the two scenarios, it can be concluded that the difference between the two cases is only 2%. This means that it is possible to increase the number of services and devices in the scenarios without generating much higher rates of CPU usage on the host system. In addition, the maximum load usage in the OT scenario occurs during the launch of the virtual machines of MECInOT, while in the Industry 4.0 scenario it occurs during the event associated with the building and deployment of the containers allocated in the industrial network. This experiment demonstrates the viability of deploying multiple scenarios on a system with limited computational resources.

5.5 Cybersecurity analysis

After evaluating the performance of the emulator, a proof of concept was conducted by carrying out cyberattacks in the Industry 4.0 scenario deployed. In this case, for the proof of concept we considered all the attacks implemented and described in Sect. 4.3. Firstly, the manipulation attack was performed, followed by the attacks related to http frames. Then the scanning network tools were run in the scenario, and the DoS attacks were implemented. Finally, the malicious device injection was run to prove its functionality. Figure 8 shows the IIoT baseline scenario designed to explain the attacks implemented in MECInOT. In this scenario, the legitimate nodes are shown with a green background, the legitimate IIoT servers are shown with a blue background, and finally the malicious node, which will introduce the network attacks in the scenario, is shown with a red background. The behaviour of the malicious node in the scenario varies in accordance with the attack that is running in the scenario.

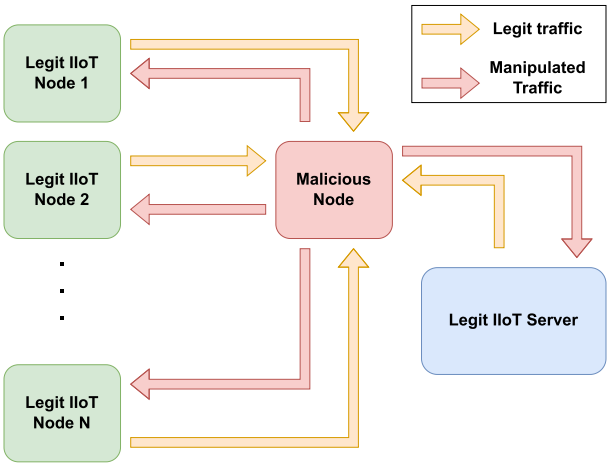

Fig. 9 Manipulation attack schema

```
s7_client_writer_1      | [*] Message send: pxCbpppfKvyNaoTqbIsD
s7_client_writer_1      | [*] Message send: rGAIJVhgSZbnikwoLTlJ
s7_client_writer_1      | [*] Message send: VrDqAiCARQCdxAJrHray
s7_client_writer_1      | [*] Message send: aQjhdJixzSRZeKBHDSsL
s7_client_writer_1      | [*] Message send: wMeCoFEtJfMaVupHZxVv
```

Fig. 10 Messages sent by the writer client

```
s7_client_reader_1      | 53435566844169441025
s7_client_reader_1      | 77410953794301182351
s7_client_reader_1      | 55736648159120147726
s7_client_reader_1      | 20972574760450636806
s7_client_reader_1      | 23851989352500968793
```

Fig. 11 Modified messages received by the reader client

5.5.1 Manipulation attack

In order to perform a successful manipulation attack, first the attacker node must execute a Man in the Middle attack using the arpspoof tool with the aim of being able to capture the traffic between the victim node and the destination. The manipulation attack scheme is shown in Fig. 9, which illustrates how the malicious node is allocated in the middle of the communication between the legitimate nodes and legitimate server, and how it returns the manipulated messages to the destination. In MECInOT the manipulation process is carried out using a Python script on the malicious node, which changes the information allocated in the data field in the corresponding IIoT protocol used in the communication. Figures 10 and 11 show how the

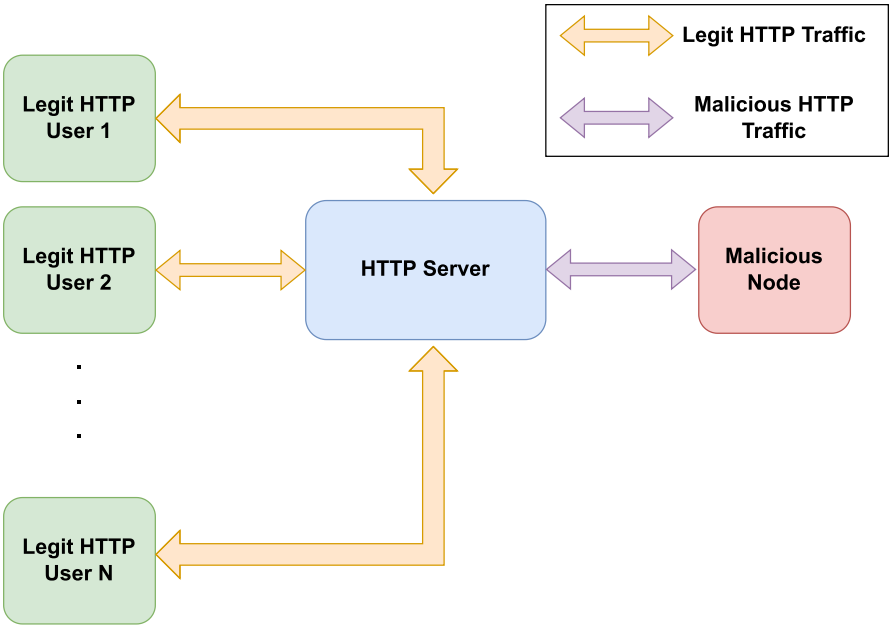

Fig. 12 HTTP applications attack schema

writer client sends strings of random characters without numbers, while the reader client receives a string of numbers from the server. This means that the attack has been successful and the results obtained are as expected.

5.5.2 HTTP application attacks

Figure 12 shows the schema followed to implement and run the brute force and payload attack against HTTP services. This schema shows how the legitimate users transmit legitimate HTTP traffic while the malicious node introduces a huge number of HTTP frames in the brute force attack and HTTP frames that carry a payload trying to exploit a possible Shellshock vulnerability allocated in the HTTP service.

The brute force attacks in MECInOT were implemented by using the Hydra tools, which is already installed on the attacker node of MECInOT. To launch this attack, there is a script which runs the tool with the corresponding parameters to indicated in the script and the dictionary used by the nmap tool, which is in charge of the enumeration of the services and passwords. Figure 13 shows a screen capture of the traffic sniffed to check whether the attack has been carried out successfully against a web server with the IP address 10.0.0.1, showing how multiple and continuous HTTP login petitions are sent to the login form.

In order to launch the Shellshock payload, a python script is used. This script facilitates the sending of the HTTP frames with the payload to gain access to the device with this vulnerability, whose IP address is 10.0.0.1 in the MEC topology.

69	19.671273706	172.18.0.3	10.0.0.1	HTTP	35782	8888	138 GET / HTTP/1.0
70	19.671274538	172.18.0.3	10.0.0.1	HTTP	35780	8888	138 GET / HTTP/1.0
71	19.671335599	172.18.0.3	10.0.0.1	HTTP	35794	8888	138 GET / HTTP/1.0
72	19.671339080	172.18.0.3	10.0.0.1	HTTP	35788	8888	138 GET / HTTP/1.0
73	19.671355335	172.18.0.3	10.0.0.1	HTTP	35786	8888	138 GET / HTTP/1.0
74	19.671668308	172.18.0.3	10.0.0.1	HTTP	35784	8888	138 GET / HTTP/1.0
81	19.672540762	172.18.0.3	10.0.0.1	HTTP	35792	8888	138 GET / HTTP/1.0
145	19.779296242	172.18.0.3	10.0.0.1	HTTP	35816	8888	422 POST / HTTP/1.0 (application/x-www-form-urlencoded)
146	19.779413401	172.18.0.3	10.0.0.1	HTTP	35818	8888	412 POST / HTTP/1.0 (application/x-www-form-urlencoded)
148	19.779446283	172.18.0.3	10.0.0.1	HTTP	35820	8888	424 POST / HTTP/1.0 (application/x-www-form-urlencoded)
153	19.779581718	172.18.0.3	10.0.0.1	HTTP	35822	8888	431 POST / HTTP/1.0 (application/x-www-form-urlencoded)
157	19.779806056	172.18.0.3	10.0.0.1	HTTP	35824	8888	421 POST / HTTP/1.0 (application/x-www-form-urlencoded)
212	19.884188042	172.18.0.3	10.0.0.1	HTTP	35826	8888	416 POST / HTTP/1.0 (application/x-www-form-urlencoded)
222	19.884502609	172.18.0.3	10.0.0.1	HTTP	35838	8888	420 POST / HTTP/1.0 (application/x-www-form-urlencoded)
224	19.895653705	172.18.0.3	10.0.0.1	HTTP	35832	8888	138 GET / HTTP/1.0
225	19.895653706	172.18.0.3	10.0.0.1	HTTP	35830	8888	138 GET / HTTP/1.0
226	19.895742890	172.18.0.3	10.0.0.1	HTTP	35828	8888	138 GET / HTTP/1.0
232	19.896720904	172.18.0.3	10.0.0.1	HTTP	35834	8888	138 GET / HTTP/1.0
256	19.906306302	172.18.0.3	10.0.0.1	HTTP	35836	8888	138 GET / HTTP/1.0
277	19.998141559	172.18.0.3	10.0.0.1	HTTP	35840	8888	408 POST / HTTP/1.0 (application/x-www-form-urlencoded)

Fig. 13 Traffic generated by the brute force attack

```
▸ Frame 821990: 262 bytes on wire (2096 bits), 262 bytes captured (2096 bits) on interface br-b099ab7e201f, id 0
▸ Ethernet II, Src: 02:42:ac:12:00:03 (02:42:ac:12:00:03), Dst: 02:42:32:25:c6:cf (02:42:32:25:c6:cf)
▸ Internet Protocol Version 4, Src: 172.18.0.3, Dst: 10.0.0.1
▸ Transmission Control Protocol, Src Port: 36094, Dst Port: 8888, Seq: 1, Ack: 1, Len: 196
▾ Hypertext Transfer Protocol
  ▸ GET /cgi/test HTTP/1.1\r\n
    Host: 10.0.0.1:8888\r\n
    Accept-Encoding: identity\r\n
    Content-type: application/x-www-form-urlencoded\r\n
    User-Agent: () { ignored;};/bin/bash -i >& /dev/tcp/localhost/8888 0>&1\r\n
    \r\n
    [Full request URI: http://10.0.0.1:8888/cgi/test]
    [HTTP request 1/1]
    [Response in frame: 821994]
```

Fig. 14 HTTP frame with malicious payload in the User-Agent field

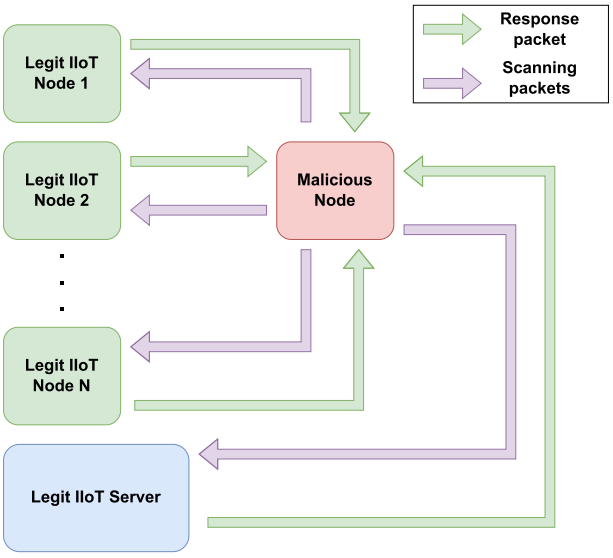

Fig. 15 Network scanning schema

487	41.591312627	172.18.0.3	10.0.0.3	TCP	51882	3560	58 51882 → 3560 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
488	41.591320534	10.0.0.2	172.18.0.3	TCP	4047	51882	54 4047 → 51882 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
489	41.591323700	10.0.0.3	172.18.0.3	TCP	8148	51882	54 8148 → 51882 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
490	41.591325634	10.0.0.1	172.18.0.3	TCP	4047	51882	54 4047 → 51882 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
491	41.591324535	172.18.0.3	10.0.0.1	TCP	51882	6473	58 51882 → 6473 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
492	41.591336696	172.18.0.3	10.0.0.2	TCP	51882	3560	58 51882 → 3560 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
493	41.591348109	172.18.0.3	10.0.0.3	TCP	51882	8588	58 51882 → 8588 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
494	41.591360343	172.18.0.3	10.0.0.1	TCP	51882	3560	58 51882 → 3560 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
495	41.591372212	172.18.0.3	10.0.0.2	TCP	51882	8588	58 51882 → 8588 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
496	41.591383592	172.18.0.3	10.0.0.3	TCP	51882	1030	58 51882 → 1030 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
497	41.591395579	172.18.0.3	10.0.0.1	TCP	51882	8588	58 51882 → 8588 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
498	41.591412757	172.18.0.3	10.0.0.2	TCP	51882	1030	58 51882 → 1030 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
499	41.591427213	172.18.0.3	10.0.0.3	TCP	51882	7326	58 51882 → 7326 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
500	41.591439569	172.18.0.3	10.0.0.1	TCP	51882	1030	58 51882 → 1030 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
501	41.591451208	172.18.0.3	10.0.0.2	TCP	51882	7326	58 51882 → 7326 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
502	41.591463788	172.18.0.3	10.0.0.3	TCP	51882	2805	58 51882 → 2805 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
503	41.591501476	10.0.0.2	172.18.0.3	TCP	8148	51882	54 8148 → 51882 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
504	41.591505295	10.0.0.3	172.18.0.3	TCP	4756	51882	54 4756 → 51882 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
505	41.591507248	10.0.0.1	172.18.0.3	TCP	8148	51882	54 8148 → 51882 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
506	41.591509541	10.0.0.2	172.18.0.3	TCP	4756	51882	54 4756 → 51882 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Fig. 16 Traffic generated by network scanning

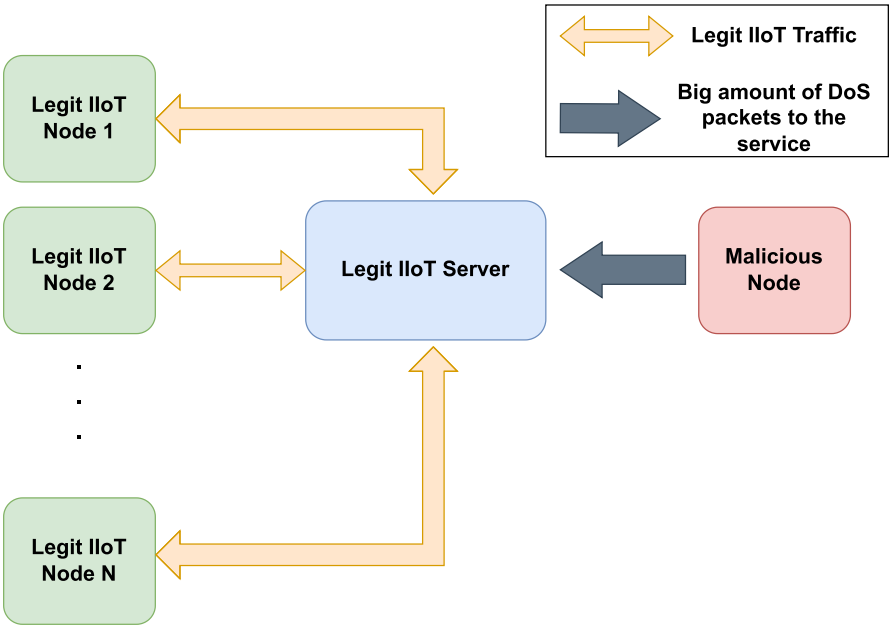

Fig. 17 DoS attack schema

Figure 14 shows an example of an HTTP packet sniffed that contains the payload in the User-Agent field.

5.5.3 Network scanning

Network scanning allows attackers to discover active nodes and servers that are running at that moment. Also, the attacker receives additional information from these devices, such as the IP address, open ports or services running. Figure 15 shows the schema for running this attack in the scenarios. The malicious node sends the scanner packets to every possible device running in the MEC-IIoT network and receives the response of each legitimate node in the network.

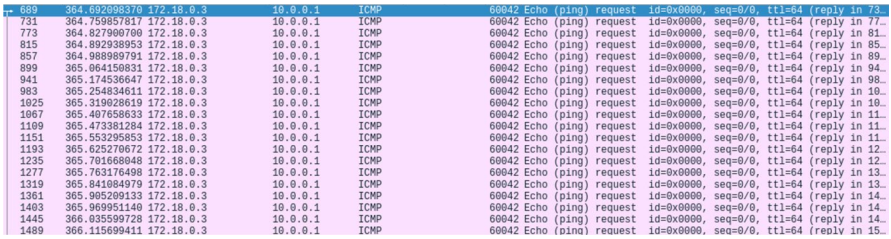

Fig. 18 Ping of Death malicious traffic

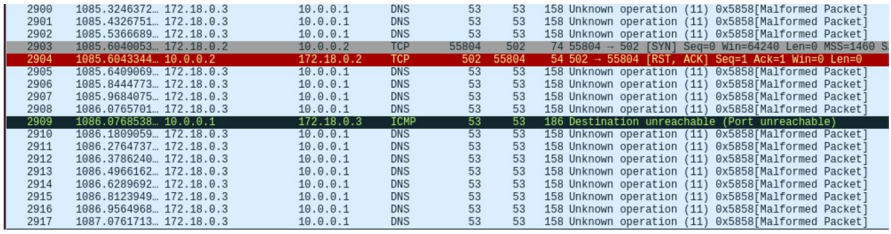

Fig. 19 Teardrop attack traffic generated

For the scanning functionality and implementation for the attacker node in MECInOT, a script was developed that automatises this task by using Python and Python-nmap modules. Figure 16 shows an example of the scanning traffic generated by the script that tries to scan the services running on the MEC servers in the scenario.

5.5.4 DoS attacks

The DoS attacks tested in this analysis are the generic ones mentioned in Sect. 4.3, and the schema DoS attack against a legitimate IIoT server in the scenario by a malicious node is shown in Fig. 17.

To run them, a Python script that allows launching the Ping of Death attack and the Teardrop attack with DNS packets is included. Figures 18 and 19 show the attacks that are launched against the edge server in the MEC topology whose IP address is 10.0.0.1. In these figures it is possible to identify the packets that are generated by the scripts, and which allow the generation malicious DoS traffic in the scenarios.

5.5.5 Malicious device injection

To evaluate the MQTT malicious device injection attack, the following devices are used: an IoT gateway with a Mosquitto broker running, two IoT devices in an IIoT topology which send temperature data to two different topics whose definitions are 'topic1' and 'topic2', and the malicious IoT device run by the attacker. Figure. 20

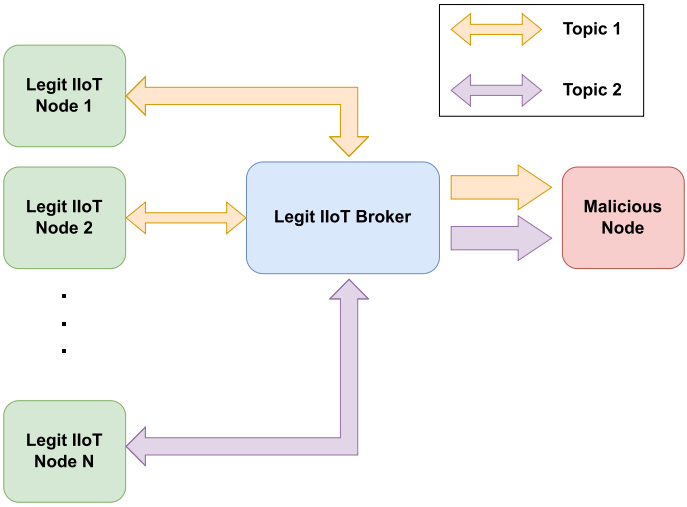

Fig. 20 Malicious device injection schema

shows the attack schema implemented in the scenario to run this attack with the devices described above.

Listing 1 shows how the malicious IoT device is implemented and how the device tries to subscribe to the # topic. If the broker only has the default configuration, the malicious IoT device will receive all the messages on any topic handled by the broker. Figure 21 shows how the malicious device receives the message from 'topic1' from the first IoT device and 'topic2' from the second one.

Under these circumstances, it can be concluded that this analysis shows that the malicious data generated in the scenario is suitable for use in the creation of new applications using big data and ML techniques, or simply for its analysis (Fig. 21).

```
cadena@cadena-VirtualBox:~/Entorno_Doc/IoT/mqtt$ python3 malicious.py
Connected with result code 0
temp1:25
temp2:29
temp1:42
temp2:38
temp1:44
temp2:45
temp1:41
temp2:45
temp1:25
temp2:26
temp1:29
temp2:26
```

Fig. 21 Messages received on other topics by malicious device

Listing 1 Malicious MQTT device implementation

```
broker_ip = "192.168.1.65"
port = 1883
mqtt_client = mqtt.Client()
mqtt_client.on_connect() = on_connect
mqtt_client.on_message() = on_message
mqtt_client.connect(broker_ip,port,60)
mqtt_client.subscribe('#')
mqtt_client.loop_forever()
```

6 IDS based on ML algorithms

In this section, we describe the methodology and workflow followed to develop the IDS based on ML algorithms. In addition, we present an analysis of the results obtained from the algorithms selected.

6.1 Methodology

The workflow for the development and deployment of the smart IDS is shown in Fig. 22.

Network Deployment. The OT scenario described in Sect. 5.2 is deployed for the IDS development. The way in which the OT services are distributed in the IIoT topology and on the MEC edge servers is shown in Fig. 23.

Data Extraction. The raw traffic data are extracted once the scenario and attacks are running. For the collection of network data, Wireshark is used on the SCADA node. Finally, we analyse these network data to extract features considered for the

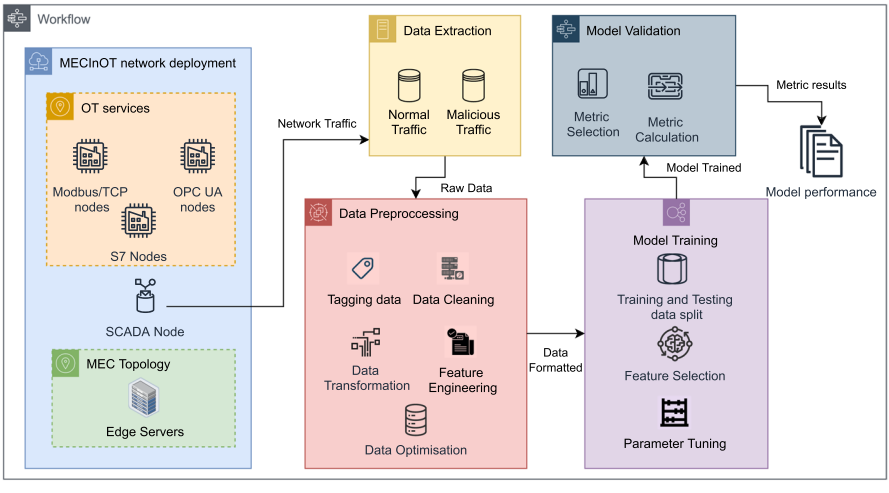

Fig. 22 Workflow of the experimentation

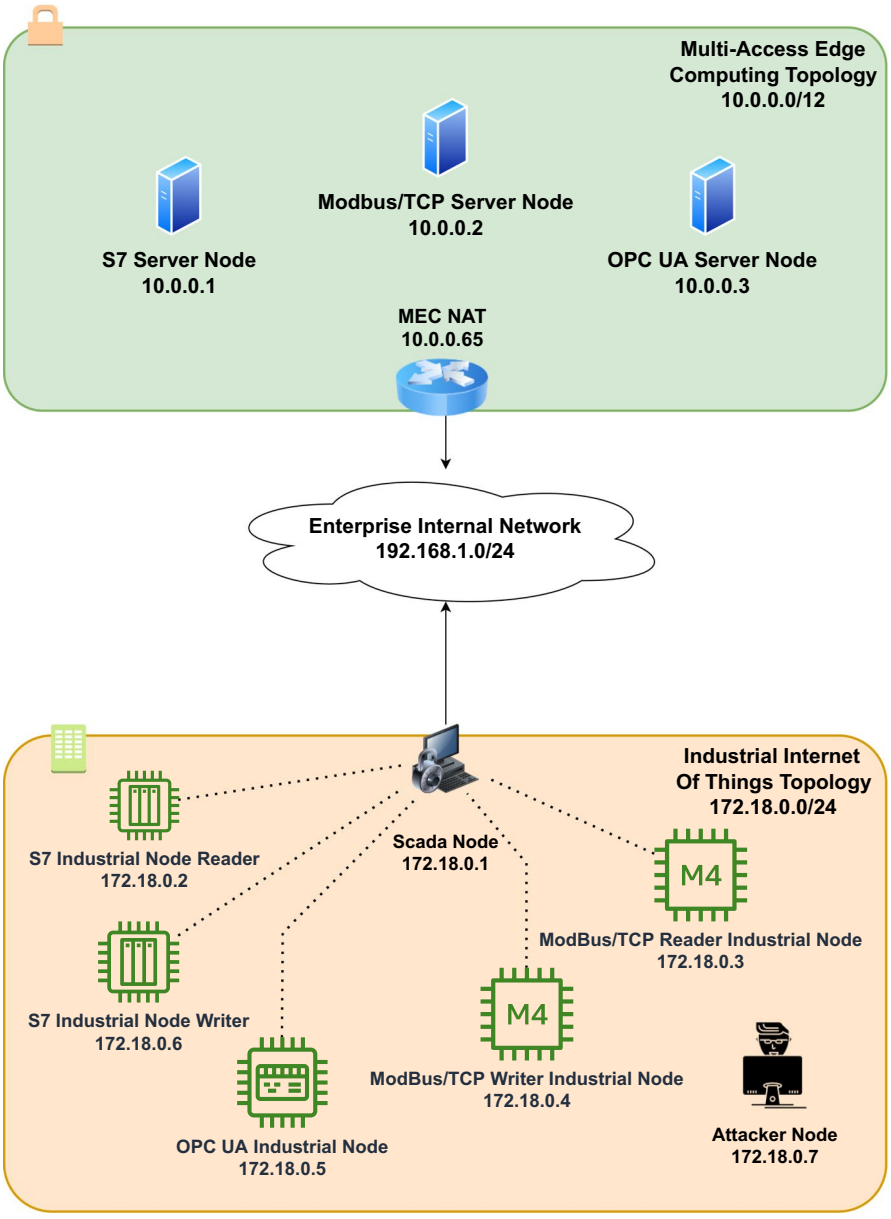

Fig. 23 IIoT experimentation scenario

input of the ML models in the following steps. The features extracted from the network packets are shown in Table 2.

Data Preprocessing. In this stage, the data extracted are adapted in order to be used with the algorithms. Firstly, the raw data have to be tagged into the

Table 2 Features extracted from the network data extraction

Feature	Description	Type
Packet number	Indicates the number of a packet. It is used as index in the dataset	Numerical
Time	Represents the specific second in which the packet was captured during the extraction	Numerical
Source	IP host source of the packet	Categorical
Destination	IP host destination of the packet	Categorical
Protocol	Protocol that transports the information of the packet	Categorical
srcPort	Source port used to send the packet	Numerical
dstPort	Destination port used to receive the packet	Numerical
Length	Total length in bytes of the packet	Numerical
Info	Additional information of the data that transport the packet captured	Categorical
tcp_Flags	Indicates the states of the flags of the TCP protocol in hexadecimal encode	Categorical
tcp_WindowSize	Indicates the size of the window of the TCP protocol	Numerical
Delay	Indicates in seconds, the difference in time between two packets of the same communication	Numerical
OPC_Message_Size	Message length of the OPC UA protocol	Numerical
OPC_Sec.Requ_ID	Indicates the ID of the OPC UA communication that responds	Numerical
MessageType	Type of message indicated in the OPC UA protocol	Categorical
Message_String	Content of the message transported in the OPC UA protocol	Categorical
S7_ROSCTR	Kind of data that the S7 protocol transports	Categorical
S7_Data_Length	Size of the S7 protocol data	Numerical
S7_Function	Function performed by the S7 packet	Categorical
S7_Message	S7 data transported in the packet encoded in base64	Categorical
S7_Error_Code	Error code from S7 protocol communication	Numerical
Mod_Length	Indicates the length of the Modbus frame	Numerical
Mod_Transa_ID	Shows the Modbus transaction id	Numerical
Mod_Function	Indicates Modbus action at destination	Categorical
Mod_Requ_Frame	Number of Modbus packet that corresponds to the request	Numerical
Mod_Req_Delay	Time between Modbus requests	Numerical

Table 2 (continued)

Feature	Description	Type
Packet number	Indicates the number of a packet. It is used as index in the dataset	Numerical
Mod_Data	Modbus message transported	Categorical
Http_user	Shows the content of User-Agent from HTTP frame	Categorical
Http_data	Shows the data transported in the HTTP frame	Categorical

Table 3 Categories, tags associated and the number of packets for each one

Category	Tag(s)	Packets
Normal traffic	final_clean	14634
HTTP attacks	brute_http, payload_user_agent	4320
DoS attacks	ping_of_death_dos, tcp_flood_dos	5533
Scanner	scanner_ack, scanner_fin, scanner_tcp, scanner_udp, scanner_xmas, scanner_null	5709
Manipulation attack	manipulation	3003

corresponding category, with the aim of enabling the models to classify them correctly. The categories tagged and the number of each one are detailed in Table 3. Moreover, it is necessary to study them to determine which techniques to apply to the data and which features are most relevant. For this purpose, the feature selection is performed with Extremely Random Forest [47]. In addition, some features are combined and retaught to receive greater importance during the training of the models.

Model Training. Once the data are ready to be used with the algorithms, namely DT, RF, NB and SVM Linear+SGD, it is necessary to distribute the data in different dataframes. These dataframes hold benign traffic and one type of attack or multiple attacks. In addition, the dataframes are divided up into 70%, which uses the K-Fold Cross-Validation technique, dividing the data into five subsets to obtain the best model, performing an experiment per subset, while the 30% of the dataset remaining is dedicated to the validation stage, which allows us to identify the performance of the model when new data are introduced after the training.

Model Validation. When the training stage is finished, it is time to validate the predictions of the final model. With the aim of evaluating whether the model correctly predicts the testing dataset, the following metrics are used: Accuracy, Precision, Recall, F1 Score and Training Time. All of these except the last one use the information of the confusion matrix, which is constructed with the number of True Positives (TP), True Negatives (TN), False Negatives (FN) and False Positives (FP) [48].

Thus, **Accuracy** metric score shows the percentage of predictions made correctly, although this metric does not explain the performance of a model when a class-imbalanced dataset is used as in our experiment. For this reason, in our analysis of the experiment Recall, Precision and F1 Score are also included. The **Recall** metric score shows the percentage that the model detects for which the packet belongs to attacks but the classification is not correct. A high **Precision** metric score means that the model does not detect many FP. The **F1 Score** shows the balance between these 2 scores. The Training Time metric is also included and it shows how much time it costs to train a model with the technique used to develop the model.

Table 4 Performance results for the models

Attack	Algorithm	Precision	Recall	F1 Score	Accuracy	Training Time (sec)
Packet manipulation	DT	0.94	0.96	0.95	0.9791	0.0152
	RF	0.96	0.96	0.96	0.9833	0.4111
	GaussianNB	0.95	0.93	0.94	0.9775	0.0092
	SVM Linear+SGD	0.94	0.91	0.92	0.9145	0.0181
Scanner	DT	0.92	0.99	0.94	0.9992	0.0182
	RF	0.92	0.99	0.94	0.9992	0.5106
	GaussianNB	0.91	0.96	0.92	0.9991	0.0153
	SVM Linear+SGD	0.91	0.96	0.92	0.9990	0.1564
DoS	DT	1	0.97	0.98	0.9994	0.0079
	RF	1	0.97	0.98	0.9994	0.2750
	GaussianNB	1	0.79	0.84	0.9962	0.0088
	SVM Linear+SGD	1	0.79	0.84	0.9962	0.0365
HTTP application	DT	1	1	1	1	0.0079
	RF	1	1	1	1	0.2750
	GaussianNB	1	1	1	1	0.0088
	SVM Linear+SGD	1	1	1	1	0.0136
Mixed	DT	0.99	0.99	0.99	0.9993	0.0660
	RF	0.99	0.99	0.99	0.9991	0.8001
	GaussianNB	0.93	0.97	0.94	0.9925	0.0313
	SVM Linear+SGD	0.52	0.57	0.5	0.9663	1.5971

6.2 Results

Table 4 shows the results for each metric and algorithm used and for each attack considered for detection by the classification model. The analysis is divided into the different attacks implemented and introduced into the dataset, so that the performance of each scenario can be evaluated.

Packet manipulation attack. This type of attack is normally difficult for classifiers to detect correctly without having a good definition of the features. The results returned by the algorithms show that the best one for detecting this attack is RF, since it achieves a score of 96% in the metrics of Precision, Recall and F1 Score, and 98.33% in Accuracy. However, RF has the worst result in terms of Training Time, and, looking at the results returned by the DT algorithm, it is possible to reduce training time without significantly sacrificing performance, as the differences in the key metrics are around 1–2 %. In addition, when it comes to obtaining better Precision than Recall results, NB is an attractive option, but it has a worse F1 score than DT. Finally, the results returned by SVM Linear+SGD are significantly worse than those of the rest of the algorithms in the study.

Scanner attacks. For the scanners that can be run in the MEC-IIoT scenarios, DT and RF return the same results for the metrics selected. Specifically, the results are 92%, 99%, 94% and 99% for the Precision, Recall, F1 Score and Accuracy metrics, respectively. The training time metric shows that DT is faster than RF, and therefore it is the best algorithm to use in this case. With respect to the NB and SVM Linear+SGD algorithms, they obtain worse results than the algorithms mentioned above. Both NB and SVM Linear+SGD obatin the same results, with 91% in Precision, 96% in Recall and 92% in F1 Score, with the only difference being 0.01% in the Accuracy metric. As can be seen, the difference in the results between DT/RF and NB/SVM Linear+SGD are not very significant.

DoS attacks. In this scenario it is possible to identify two groups of results: the first one, with 100% in Precision, 97% in Recall, 98% in F1 Score and 99.94% in Accuracy, is the one obtained by the algorithms DT and RF. Therefore, the best-performing one in this group, and in general, is DT as it is faster to train than RF. Specifically, DT takes 0.0079 s to be trained, instead of the 0.2750 s that RF needs to finish the training model. The second group returns worse results in general than the first one, with 100%, 79%, 84% and 99.62% in the Precision, Recall, F1 Score and Accuracy metrics, respectively.

HTTP application attacks. The results for the HTTP attacks implemented in the scenario show that all the algorithms achieve 100% in for all the metrics. The reason for this is that the parameters considered for the models are able to detect the attacks easily thanks to the information in the HTTP frames for payload attacks, and the time delay between packets for brute force attacks. Therefore the selection of the best algorithm should be based on another metric, such as the resources needed to run the model or the training time of the models, depending on the needs of each scenario in which the IDS will be implemented. When using the training time it is possible to determine that the best algorithm for this type of attacks is DT, with 0.0079 s.

Mixed attacks. This case is the most realistic scenario that it is possible to find in real Industry 4.0 factories when they are under attack from multiple attackers at the same time. Under these circumstances, the results obtained in this experiment are crucial in order to select the algorithm to be used for the IDS implementation. The best results are achieved by the DT and RF algorithms, which obtain 99% for every metric. The main difference between them is the training time, with a better result being achieved by the DT algorithm. In addition, it is important to highlight the poor results returned by the SVM Linear+SGD algorithm for the Precision, Recall and F1 Score metrics, with these being around 50%. This is particularly curious when the Accuracy metric shows 99.63%, which means that SVM Linear+SGD does not correctly classify the packets and is probably grouping most of them in to a single class.

Finally, an analysis of the metrics shows that the best options for all the testing scenarios are DT and RF, since both of them return similar results. As has been mentioned during the analysis of each case, the main difference between the two solutions is in the Training Time metric. This metric gives an idea of how much time it takes to train the model with a specific parameter, and how it could scale when the dataset size increases. Furthermore, NB produces very similar results

to DT with a difference in performance of around 1-6 % in the Packet Manipulation, Scanner, HTTP and Mixed attacks. However, the results obtained in the DoS attacks are significantly worse than the DT and RF algorithms. Lastly, the worst ML algorithm included in the analysis is SVM Linear+SGD, with a performance similar to NB in the Packet Manipulation, Scanning, DoS and HTTP attacks. However, it obtains the worst performance for the Mixed attacks and in all cases returns the longest Training Time.

7 Conclusion

In this work, MCEInOT, a cybersecurity-oriented emulator for the deployment of MEC-IIoT topologies for experimentation in this context has been presented. It provides cybersecurity researchers with different tools for carrying out network and application attacks in any scenario deployed with the emulator, supporting IIoT, IoT and IT protocol-based applications. In addition, it has been shown that our proposal can be used to extract data from network attacks made on the network on which the experimentation scenario is deployed, and then use these data to train different ML algorithms to be deployed as an IDS for the MEC-IIoT topology. The experiment has allowed us to evaluate some ML algorithms for classification purposes, namely DT, RF, NB and SVM Linear+SGD. Although they provide quite similar performances in some situations, the DT and RF achieve the best general results, with the training time metric showing that DT obtains better results than RF.

8 Future work

In this section, we describe several future projects that could improve the emulator, as well as some lines for additional research that could derive from this experiment.

- **Optimisation of the container images.** It would be highly beneficial to reduce the size of the containers in the scenario in order to improve the speed and resource consumption during the deployment of the topology of the application nodes.
- **Unification of the emulator on a single virtual machine.** In order to properly use the emulator on a single computer, it is necessary to run two virtual machines and interconnect them using a private network. This produces an overuse of computational and network resources that could be avoided if the two virtual machines were joined into a single one.
- **Extend experiments with other ML and Deep Learning (DL) techniques.** In this work, a study of 4 ML algorithms for the development of an IDS was carried out. However, there is the option of testing additional ML techniques such as kernel-based algorithms or boosting-tree algorithms. Even some lightweight DL approaches could be considered for integration into MEC-IIoT environments.

- **Study of implementation of the IDS in the scenario.** Once it has been decided which ML algorithm to select to implement the IDS, it is necessary to deploy it in the scenario and study its performance when the attacks are running. Therefore, it would be useful to analyse the possible implementations of the IDS given the possibilities offered by the inclusion of an MEC topology in the IIoT scenario. In addition, various IDS architecture proposals can be implemented, using MEC as the principal component to control the traffic or deploy the smart IDS on each device in order to detect possible anomalies individually.
- **Implementing new network functionalities.** MECInOT is mainly an emulator oriented for use in the cybersecurity field, so it would be interesting to implement other network functionalities such as firewalls, or topologies with a cyberdefence architecture in mind near the implementation of a Demilitariesed Zone (DMZ), in order to improve the quality and the possibilities during experimentation.
- **Introduction of new applications into scenarios.** Thanks to the possibilities that openLEON provides when deploying different MEC topologies, it would be interesting to offer the option of implementing new emerging technologies oriented to wards security and privacy. One example would be the smart contracts based on blockchain to preserve the privacy and integrity of the data.
- **Inclusion of specific attacks for the different protocols.** The attacks included and described in MECInOT are specifically designed for carrying out generic network cyberattacks in the scenarios that the emulator can deploy. However, it would be useful to add new types of attacks that are focused on exploiting vulnerabilities found on OT and IoT devices or in their protocols. In addition, these attacks should consider not only IIoT vulnerabilities but also those associated with the services and the virtualisation functions in the MEC paradigm [49].

Author Contributions Sergio Ruiz-Villafranca conceived and designed the emulator and experiments, performed the experiments, analysed the data, performed the computation work, prepared figures and or tables, authored or reviewed drafts of the paper, and approved the final draft. Javier Carrillo-Mondejar conceived and designed the experiments, performed the experiments, prepared figures and or tables, authored or reviewed drafts of the paper, and approved the final draft. Juan Manuel Castelo Gómez conceived and designed the emulator, analysed the data, performed the computation work, prepared figures and or tables, authored or reviewed drafts of the paper, and approved the final draft. Jose Roldan-Gomez analysed the data, performed the computation work, authored or reviewed drafts of the paper, and approved the final draft.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work was supported by the University of Castilla-La Mancha under the predoctoral contract PI001482 and the postdoctoral contract 2021-POST-20518, both financed by the European Social Fund Plus (FSE+), by the JCCM under the project SBPLY/21/180501/000195, and by the Spanish Education, Culture and Sports Ministry under grants FPU 17/03105. Also, this work is part of the R &D project PID2021-123627OB-C52, funded by the MCIN and the European Regional Development Fund: “a way of making Europe”.

Data availability statement The data used in this work are available in the following link: https://data.mendeley.com/datasets/xstyjwrc5r/draft?a=440538fd-e139-4e06-8885-107785f51807.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical approval Not applicable

References

1. Ivanov D, Tang C, Dolgui A, Battini D, Das A (2020) Researchers' perspectives on industry 4.0: multi-disciplinary analysis and opportunities for operations management. Int J Product Res 201:1–24. https://doi.org/10.1080/00207543.2020.1798035
2. Maddikunta PKR, Pham Q-V, Deepa N, Dev K, Gadekallu TR, Ruby R, Liyanage M (2022) Industry 5.0: A survey on enabling technologies and potential applications. J Indust Inform Integrat 26:100257. https://doi.org/10.1016/j.jii.2021.100257
3. Xu X, Lu Y, Vogel-Heuser B, Wang L (2021) Industry 4.0 and industry 5.0-inception, conception and perception. J Manufact Syst 61:530–535. https://doi.org/10.1016/j.jmsy.2021.10.006
4. Filali A, Abouaomar A, Cherkaoui S, Kobbane A, Guizani M (2020) Multi-access edge computing: A survey. IEEE Access 8:197017–197046
5. Dhirani LL, Armstrong E, Newe T (2021) Industrial iot, cyber threats, and standards landscape: Evaluation and roadmap. Sensors 21(11):3901
6. Iaiani M, Tugnoli A, Bonvicini S, Cozzani V (2021) Analysis of cybersecurity-related incidents in the process industry. Reliab Eng Syst Safety 209:107485. https://doi.org/10.1016/j.ress.2021.107485
7. Shen M, Liu A, Huang G, Xiong NN, Lu H (2021) Attdc: an active and traceable trust data collection scheme for industrial security in smart cities. IEEE Int Things J 8(8):6437–6453. https://doi.org/10.1109/JIOT.2021.3049173
8. Chander B, Pal S, De D, Buyya R (2022). In: De D, Buyya R, Pal S (eds) Artificial intelligence-based internet of things for industry 5.0. Springer, Cham, pp 3–45
9. Fiandrino C, Pizarro A, Mateo P, Andrés Ramiro C, Ludant N, Widmer J (2019) Openleon: an end-to-end emulation platform from the edge data center to the mobile user. Comput Commun 148:17–26. https://doi.org/10.1016/j.comcom.2019.08.024
10. Auliva RS, Sheu R-K, Liang D, Wang W-J (2018) Iiot testbed: A dds-based emulation tool for industrial iot applications. In: 2018 International Conference on System Science and Engeering (ICSSE), pp. 1–4. https://doi.org/10.1109/ICSSE.2018.8520091
11. Luo G, Chen Z, Mohammed BO (2022) A systematic literature review of intrusion detection systems in the cloud-based IoT environments. Concurr Computat Pract Exp 34(10):6822. https://doi.org/10.1002/cpe.6822
12. Moysis S, Zacharias G, Demetris T, George P, Marios D D (2020) Fogify: A fog computing emulation framework. In: Proceedings of the 5th ACM/IEEE Symposium on Edge Computing. SEC '20. Association for Computing Machinery. New York, NY, USA

13. Coutinho A, Greve F, Prazeres C, Cardoso J (2018) Fogbed: A rapid-prototyping emulation environment for fog computing. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–7. https://doi.org/10.1109/ICC.2018.8423003
14. Rodrigues TK, Liu J, Kato N (2021) Application of cybertwin for offloading in mobile multiaccess edge computing for 6g networks. IEEE Int Things J 8(22):16231–16242. https://doi.org/10.1109/JIOT.2021.3095308
15. Liu J, Li Q, Cao R, Tang W, Qiu G (2020) Mininet: an extremely lightweight convolutional neural network for real-time unsupervised monocular depth estimation. ISPRS J Photog Remote Sens 166:255–267
16. Kreutz D, Ramos FM, Verissimo PE, Rothenberg CE, Azodolmolky S, Uhlig S (2014) Software-defined networking: a comprehensive survey. Proceed IEEE 103(1):14–76
17. Pham Q-V, Fang F, Ha VN, Piran MJ, Le M, Le LB, Hwang W-J, Ding Z (2020) A survey of multi-access edge computing in 5g and beyond: fundamentals, technology integration, and state-of-the-art. IEEE Access 8:116974–117017
18. Liyanage M, Porambage P, Ding AY (2018) Five driving forces of multi-access edge computing. arXiv preprint arXiv:1810.00827
19. Mahesh B (2020) Machine learning algorithms-a review. Int J Sci Res (IJSR) 9:381–386
20. Roldán J, Boubeta-Puig J, Luis Martínez J, Ortiz G (2020) Integrating complex event processing and machine learning: An intelligent architecture for detecting iot security attacks. Expert Syst Appl 149:113251. https://doi.org/10.1016/j.eswa.2020.113251
21. Suthishni DNP, Kumar KSS (2022) A Review on Machine Learning based Security Approaches in Intrusion Detection System. In: 2022 9th International Conference on Computing for Sustainable Global Development (INDIACom), pp. 341–348. https://doi.org/10.23919/INDIACom54597.2022.9763261
22. Mohammed M, Khan MB, Bashier EBM (2016) Machine learning: algorithms and applications. CRC Press
23. Azuaje F, Witten IEF (2006) Witten ih, frank e: data mining: practical machine learning tools and techniques. Biomed Eng Online 5:1–2
24. Sarker IH (2021) Machine learning: algorithms, real-world applications and research directions. SN Comput Sci 2(3):1–21
25. Salzberg SL (1994) C45: programs for machine learning by j ross quinlan. Mach Learn 16(3):235–240. https://doi.org/10.1007/BF00993309
26. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V et al (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830
27. Breiman L (2001) Random forests. Mach Learn 45:5–32. https://doi.org/10.1023/A:1010950718922
28. John GH, Langley P (1995) Estimating continuous distributions in bayesian classifiers. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. UAI'95, pp. 338–345. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
29. Ruder S (2016) An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747
30. Mohammadi M, Rashid TA, Karim SHT, Aldalwie AHM, Tho QT, Bidaki M, Rahmani AM, Hosseinzadeh M (2021) A comprehensive survey and taxonomy of the svm-based intrusion detection systems. J Net Comput Appl 178:102983. https://doi.org/10.1016/j.jnca.2021.102983
31. Smys S, Chen JIZ, Shakya S (2020) Survey on neural network architectures with deep learning. J Soft Comput Parad (JSCP) 2(03):186–194
32. Shwartz-Ziv R, Armon A (2022) Tabular data: deep learning is not all you need. Inform Fus 81:84–90. https://doi.org/10.1016/j.inffus.2021.11.011
33. Roveri M (2023) Is tiny deep learning the new deep learning? Computational Intelligence and data analytics. Springer, London, pp 23–39
34. Mishra B, Kertesz A (2020) The use of mqtt in m2m and iot systems: a survey. IEEE Access 8:201071–201086
35. Silva D, Carvalho LI, Soares J, Sofia RC (2021) A performance analysis of internet of things networking protocols: evaluating mqtt, coap, opc ua. Appl Sci 11(11):4879
36. Goldenberg N, Wool A (2013) Accurate modeling of modbus/tcp for intrusion detection in scada systems. Int J Crit Infrast Protect 6(2):63–75. https://doi.org/10.1016/j.ijcip.2013.05.001
37. Hui H, McLaughlin K, Sezer S (2021) Vulnerability analysis of s7 plcs: manipulating the security mechanism. Int J Crit Infrast Protect 35:100470. https://doi.org/10.1016/j.ijcip.2021.100470

38. Lederer S, Müller C, Timmerer C (2012) Dynamic adaptive streaming over http dataset. In: Proceedings of the 3rd Multimedia Systems Conference, pp. 89–94
39. Mary C (2015) Shellshock attack on linux systems-bash. Int Res J Eng Technol 2(8):1322–1325
40. Abdollahi A, Fathi M (2020) An intrusion detection system on ping of death attacks in iot networks. Wirel Person Commun 112(4):2057–2070
41. Thomas DR, Clayton R, Beresford AR (2017) 1000 days of udp amplification ddos attacks. In: 2017 APWG Symposium on Electronic Crime Research (eCrime), pp. 79–84. IEEE
42. Peuster M, Karl H, van Rossem S (2016) Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 148–153. https://doi.org/10.1109/NFV-SDN.2016.7919490
43. Kaur K, Singh J, Ghumman NS (2014) Mininet as software defined networking testing platform. In: International Conference on Communication, Computing & Systems (ICCCS), pp. 139–42
44. Grygorash O, Zhou Y, Jorgensen Z (2006) Minimum spanning tree based clustering algorithms. In: 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), pp. 73–81. IEEE
45. Asadollahi S, Goswami B, Sameer M (2018) Ryu controller's scalability experiment on software defined networks. In: 2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), pp. 1–5. IEEE
46. Gomez-Miguelez I, Garcia-Saavedra A, Sutton P, Serrano P, Cano C, Leith D (2016) srslte: an open-source platform for lte evolution and experimentation, pp. 25–32. https://doi.org/10.1145/2980159.2980163
47. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach learn 63(1):3–42
48. Handelman GS, Kok HK, Chandra RV, Razavi AH, Huang S, Brooks M, Lee MJ, Asadi H (2019) Peering into the black box of artificial intelligence: evaluation metrics of machine learning methods. Am J Roentgenol 212(1):38–43
49. Roman R, Lopez J, Mambo M (2018) Mobile edge computing, fog et al.: a survey and analysis of security threats and challenges. Future Generat Comput Syst 78:680–698. https://doi.org/10.1016/j.future.2016.11.009

Authors and Affiliations

Sergio Ruiz-Villafranca[1] · Javier Carrillo-Mondéjar[2] · Juan Manuel Castelo Gómez[1] · José Roldán-Gómez[3]

Javier Carrillo-Mondéjar
jcarrillo@unizar.es

Juan Manuel Castelo Gómez
juanmanuel.castelo@uclm.es

José Roldán-Gómez
roldangjose@uniovi.es

[1] University of Castilla-La Mancha, Campus Universitario s/n, Albacete 02006, Spain

[2] University of Zaragoza, Zaragoza, Spain

[3] Department of Computer Science, University of Oviedo, Gijón, Spain