Designing a Forensic-Ready Wi-Fi Access Point for the Internet of Things

Fabio Palmese, *Graduate Student Member, IEEE*, Alessandro Enrico Cesare Redondi, *Senior Member, IEEE*, and Matteo Cesana, *Senior Member, IEEE*

***Abstract*—Recent advances in the Internet of Things are leading to a proliferation of smart devices in our daily life. Having so many connected devices around us potentially introduces new witnesses that can be a reference for forensic investigations. For these reasons, IoT Forensics has become a popular research area with the goal of extracting information from IoT devices to be used as potential evidence. This work presents *Feature-Sniffer*, a framework to be installed in Wi-Fi access points with the aim of facilitating the extraction of network traffic information from IoT devices, to be later used for forensic purposes. The tool allows the on-the-fly computation of traffic features from connected IoT devices by using a simple user interface for its configuration. After presenting the tool logic and its implementation details, we present an accurate analysis of the tool computational impact on two different consumer Wi-Fi access points. Finally, we present four different IoT forensics use cases, in which network traffic features extracted with the proposed tool from consumer IoT devices are analyzed with machine learning techniques with the goal of 1) identifying the device producing the traffic; 2) recognizing the activity performed by the user; 3) detecting the user's passage through a room door; and 4) detecting and classifying user interactions with a smart speaker. We conclude the work by presenting an analysis of possible storage optimization for evidence preservation with the use of lossy compression techniques.**

***Index Terms*—Internet of Things, IoT forensics, network traffic analysis, network traffic collection.**

I. Introduction

WE ARE living in the Internet of Things era. The devices surrounding us are becoming smart and new objects, such as home assistants, have become the main characters in our daily activities. If on the one hand, such devices help us in many aspects, they can also introduce potential privacy and security threats. Being such devices always connected, they can be potentially used by attackers for collecting private information or even for controlling our home environment. At the same time, with all the devices always present in our living environment, there are additional witnesses in our daily activities that could be used for forensic purposes. The extraction of such information is the goal of a new branch of digital forensics, named IoT Forensics. The device now not only serves the user performing its operations but could be leveraged for extracting potential sources of evidence that can help to solve forensic cases, unveiling information on the device itself, the user, and the surrounding environment. IoT forensics has recently attracted the attention of the current research in all its different aspects [1]. The state-of-the-art divides IoT forensics into three different layers, depending on where the information is retrieved: 1) cloud forensics; 2) network forensics; and 3) device-level forensics. Our work focuses on the middle layer, and in particular on network traffic extracted from IoT devices to unveil information on the device and the user activities. Proposed methodologies from the literature usually require the setup of advanced collection pipelines for 1) collecting the network traffic in the network in which devices are located and 2) extracting potentially relevant characteristics from the traffic to be used for forensic tasks. The data are usually managed in the form of PCAP files and require a consistent post-processing phase for the information extraction, which makes forensic analysis usually very time-consuming. This work extends our previous study presented in [2], presenting more advanced results and extending the proposed framework functionalities. The work takes the goal of solving the main limitations of PCAP-based forensic analysis by proposing a framework, named Feature-Sniffer, to be installed in the smart home gateways and that can directly collect the main network traffic characteristics from IoT devices, avoiding splitting the pipeline into two separate phases of collection and feature extraction. Feature-Sniffer can be installed in any Wi-Fi access point supporting the OpenWrt project. The tool allows on-the-fly extraction of network traffic features, aggregating captured packets into time windows of a fixed duration and extracting statistical features on network/transport layer characteristics as well as physical layer indicators. The tool can be easily configured with a simple Web interface through which the user can totally customize the collection: it is possible to select the network interface from which to capture packets, the window duration for grouping packets, the features to compute, and several parameters to obtain a personalized output. By using the proposed framework, it is possible to save both in storage and time domains, avoiding storing the full raw traffic as PCAP files, which usually also contain nonrelevant information, and saving time in the analysis by directly producing the required features.

Manuscript received 26 May 2023; revised 31 July 2023 and 1 August 2023; accepted 4 August 2023. Date of publication 11 August 2023; date of current version 21 November 2023. This work was supported by the Next-Generation EU (Italian PNRR-M4 C2, Invest 1.3-D.D. 1551.11-10-2022, CUP MICS D43C22003120001) under Grant PE00000004. *(Corresponding author: Fabio Palmese.)*

The authors are with the Department of Electronics, Information and Bioengineering, Politecnico di Milano, 20133 Milan, Italy (e-mail: fabio.palmese@polimi.it; alessandroenrico.redondi@polimi.it; matteo.cesana@polimi.it).

Digital Object Identifier 10.1109/JIOT.2023.3304423

The contribution of this work is as follows: first, we present the Feature-Sniffer framework, focusing on its working principle as well as on its implementation details. After the presentation of the tool in all its features, we present an overview of its performance impact in modern Wi-Fi networks by showing the CPU consumption under stressed network conditions using two different consumer Wi-Fi access points. To complete the work, we show potential application cases of the tool, using it to perform four different IoT forensics activities: 1) IoT devices identification analyzing a public widely used data set containing modern common devices traffic traces; 2) human activity recognition from IoT camera encrypted traffic; 3) human passage detection with physical layer features extracted from a generic IoT device used as a source of traffic; and 4) detection and classification of human interactions with a smart speaker. The work concludes with an analysis of the data set storage occupation and presenting possible optimization techniques to reduce the data set size preserving the forensic capabilities. Feature-Sniffer source code and installation guidelines are publicly available.[1]

The remainder of this article is structured as follows: Section II presents the main related works on IoT Forensics. Section III provides an accurate description of the Feature-Sniffer tool with implementation and installation details. Section IV presents the computational impact of the tool when installed in two commercial access points and highlights the affordability of the tool in modern real-life networks. Four different IoT forensics use cases are discussed in Section V, and the corresponding experimental results obtained with the application of machine learning techniques are summarized in Section VI, including an analysis of potential storage optimization. Section VII concludes the works with some final remarks and possible future research directions.

II. Related Work

Most IoT devices in modern smart home networks communicate using encrypted traffic for both privacy and security reasons. The use of encryption, however, does not exclude the possibility of extracting information from such devices by only inspecting the corresponding network traffic. Many works in the literature revealed that it is possible to extract potentially private information from such devices using the most common network traffic sniffer and analyzer tools. Ren et al. [3] presented a deep analysis of consumer IoT devices located in two laboratories, one in the U.S. and one in the U.K. The work considers more than 80 different modern IoT devices and highlights that for most of them, it is possible to understand the activity that caused the devices to generate the traffic by only inspecting network packets. In the work presented in [4] instead, Wan et al. proposed a framework named IoTAthena capable of characterizing the IoT device activities from the raw network packets. The framework proposes two novel polynomial-time algorithms to extract device activities and has been tested on 16 different consumer devices, including smart bulbs, smart plugs, and other types. Many other works have shown that it is possible to expose information from IoT devices only referring to the network traffic they use for communication [5], [6], [7]

Advanced IoT forensic analysis pipelines usually assume as known the type of device producing the traffic (i.e., analyzing home assistant traffic for speaker detection). However, this assumption does not hold in all application cases, and there are forensic cases in which it can be crucial to know the type of device present in the environment. For this reason, several works in the current research have explored the possibility of identifying IoT devices only relying on the produced traffic, and we will do so later in this work. Sivanathan et al. [8] presented a two-way approach for successfully identifying the device producing the traffic, with a first classification based on domain names, remote ports, and cipher suites, and a second stage applying a random forest classifier to the previous stage results, adding network features, such as flow rate and volume. The authors tested the proposed approach using the network traffic from 31 IoT and non-IoT devices produced in a period of six consecutive months and obtained remarkable results with over 99.8% F1-Score. As an additional contribution, the authors publicly shared the data set in the form of PCAP files for research purposes. Different solutions for identifying the different devices from their network traffic traces have been proposed in the literature (e.g., [9] and [10]). However, all these works usually require advanced collection pipelines and long processing times, which is one of the reasons that led us to propose the framework in this work. In the same research line, Shahid et al. [11] proposed a solution based on t-distributed stochastic neighbor embedding (t-SNE) for identifying the device only relying on the first N network packets of each bidirectional flow of the device. However, in this work, the approach is very limited by the number of devices considered (only four devices are used) and should be tested in bigger networks.

Once the device type is known, either by identification or with apriori knowledge, it is possible to reason on device-specific tasks for extracting potential forensic information on the device, the user, or the surrounding environment. Among the others, many researchers focused their attention on smart home environments analyzing devices, such as home assistants [12], smart cameras [13], or even combining events from different devices to infer human activities [5], [14]. It is worth reporting here the main related works based on traffic analysis from smart cameras and smart speakers, as these types of devices are widely analyzed in the next sections of this work. The work in [13] analyzes the traffic produced by two different IP cameras to distinguish nine daily activities performed by the user in front of the camera Field of View (FoV, i.e., dressing, sporting, and reading). The authors apply six different machine learning classifiers to several statistical features extracted from the network traffic produced by the cameras during user activities. The main considered features are packet interarrival times, transmission rates, and payload lengths extracted from the traffic stored in 9000 PCAP files (one per activity). Wampler et al. [15] highlighted the possibility of detecting a motion or a scene change from

[1] https://github.com/fpalmese/feature-sniffer

encrypted traffic produced by IP cameras, regardless of the codec used by the devices. They additionally propose a graphical analysis of the network traffic, plotting several features over time, including interarrival time between packets, packet sizes, and video stream bandwidth. Finally, Li et al. [16] used two common IoT cameras from Google and Samsung to show that attackers spoofing the network traffic from such devices could easily distinguish basic activities of the user's daily life.

For what concerns Wi-Fi physical layer analysis, many works in the current state-of-the-art analyzed the correlation of the Wi-Fi channel state information (CSI) with human activity in the surrounding environment. Soto et al. [17] presented a survey on the main works using CSI features to detect vital signs from the surrounding environment: several works show that it is possible to use such features for human heart rate and respiration monitoring. However, the data-gathering pipeline in such works is usually strictly task-specific and requires proper hardware/software setups. This article, on the contrary, does not assume any specific transmission setup and allows the extraction of information from already deployed IoT devices.

Given the widespread popularity of smart speakers in modern smart homes, many research works in the last years focused on analyzing the corresponding network traffic to understand the consequences and impact of the presence of such devices on user privacy. Caputo et al. [18] showed that it is possible to detect the human presence in the environment in which a Google Home device is located, even if the user is not interacting with the device. Several works have been performed with the goal of detecting if a smart speaker is recording and transmitting sounds from the environment, even if not explicitly triggered using the proper activation words [19], [20]. Other works in the literature focused on the extraction of forensically relevant information from smart speakers with the goal of fingerprinting voice commands [21], [22], [23] or distinguishing the speaker interacting with the device [24]. However, most of the proposed models are evaluated using data produced with synthetic speech to activate the devices. Differently, in this work, we use a combination of real and synthetic speech.

To conclude the section, we report the main related work proposing IoT forensics frameworks. Meffert et al. [25] presented a framework to be installed in a centralized smart hub that allows to control and concurrently acquire the state of consumer IoT devices in a smart home environment. The work in [26] presents IoTDots, a forensic framework that processes the mobile smart applications used for controlling the IoT devices and introduces an automatic logging procedure to store potential forensic data in a centralized database, subsequently analyzed with machine learning techniques to perform forensic tasks. However, in both frameworks, the data acquisition does not consider network traffic produced by the devices, which can unveil very relevant forensic information. Liu et al. [27] presented a script to be executed in OpenWrt-based systems that allows keeping track of devices connection logs by capturing TCP SYN packets. The output is optionally transmitted periodically to external entities using emails.

III. Feature-Sniffer

A. Overview

The related work on network traffic analysis applied to IoT forensics puts in evidence the main limitation when working with IoT devices network traffic: the huge amount of data to handle. The state-of-the-art methodology usually involves the application of machine learning classification techniques to features extracted from network traffic obtained by storing the full information in the form of a PCAP file and then processing it. The storage of the raw traffic in PCAP files introduces two main limitations.

1) The amount of occupied storage becomes a problem in IoT scenarios.
2) The amount of time to process all the packets stored in the PCAP file is not trivial and introduces a consistent delay in the forensics analysis phase.

Moreover, considering the transient nature of the network traffic, the acquisition of such data requires the preparation of the collection environment to be able to capture the packets that are flowing for the communication, otherwise the information simply gets lost. This requires the hardware to be properly set for collection and to be able to store several days/months of traffic to maintain the information until it is needed. To solve these PCAP-related limitations, this work presents Feature-Sniffer, a tool that allows computing network traffic features in an online fashion on any consumer WiFi access point operated by the OpenWrt firmware, hence avoiding the cumbersome tasks of 1) dumping network traffic to PCAP files and 2) implementing ad-hoc routines for analyzing the captured traffic.

The tool is designed to be extremely user friendly so that network administrators, data scientists, or forensic investigators can easily extract and make use of a multitude of network traffic features instead of building ad-hoc traffic capture and analysis pipelines. In a nutshell, Feature-Sniffer analyzes all network packets that flow through the access point and organizes them into time windows of user-defined duration. For each time window and for each device connected to the access point (identified by its MAC address), the tool extracts and stores a set of statistical traffic features which can be selected by the user through the access point graphical user interface (GUI). Fig. 1 sketches the architecture of the proposed tool, which is composed of three main parts: 1) an application for the LuCI Web interface, developed to access all functionalities directly from the access point control panel; 2) a feature extraction engine (FEE), developed in the C programming language and easily configurable through the Web interface; and 3) A physical layer FEE (P-FEE) responsible of the collection of RSSI and CSI-based features. The next sections will explain the main components of the proposed framework in detail.

B. Envisioned Usage

We present here a small description of the envisioned usage of the tool. A network administrator wants to install an IoT forensic-ready access point on the premises of a Smart Home/Smart Building. Therefore, she sets up an access point

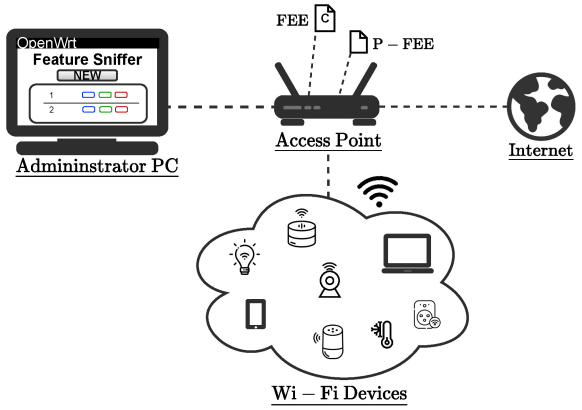

Fig. 1. Feature-Sniffer architecture. The OpenWrt control panel on the access point allows the interaction with the FEEs in order to perform IoT forensics analysis.

with Feature-Sniffer installed. If the network administrator also has a forensic background and possesses prior knowledge of the type of IoT devices present in the network or the forensics activities that might be performed in case of an anomalous event, she can create ad hoc configurations selecting only the network features that will be useful for such analysis. Otherwise, in case of no prior knowledge, a basic configuration with all possible network features selected will constitute the most informative approach and can be set to start as the access point is powered on automatically. The network administrator can, however, decide to control the forensic acquisition through the Web interface buttons to start and stop the capture processes. In case something happens and a forensic analysis is requested, the network administrator can download the Feature-Sniffer output through the interface and pass it to the investigator for proper analysis and information extraction.

C. Web Interface

The Feature-Sniffer user interface is an add-on built on top of LuCI functionalities.[2] LuCI is the standard Web user interface for OpenWrt systems. It is based on the Lua programming language, which easily allows the creation of customized extensions. As illustrated in Fig. 2, we extended LuCI with a new user interface dedicated specifically to IoT Forensics operations. The interface allows the network administrator to quickly create traffic analysis *configurations*, used to extract specific features from the network traffic flowing through the access point. The main page shows all the existing configurations with the corresponding information and control buttons (start, stop, edit, download output, delete), as well as a section to create new configurations.

Each configuration is saved in a *.cfg* file, which is made compliant with the `libconfig` configuration C library and is characterized by the following parameters.

1) *Name and Description:* Representing the configuration unique identifier, as well as a brief optional text describing the configuration purposes.
2) *Window Duration:* The duration in time of each analysis window. Traffic features are computed independently in each window.
3) *Capture Filter:* An optional string following the libpcap *pcap_filter* syntax to extract features only from the traffic of interest (e.g., only outgoing UDP traffic).
4) *Device Filter:* The user may insert here the MAC addresses of the devices to consider in the analysis. If the list is left empty, all devices associated with the access point are considered.
5) *Capture Type:* Live or from PCAP. In Live capturing mode, the packets will be captured and analyzed from one of the network interfaces of the access point. This mode requires the user to specify the interested interface as an additional mandatory parameter. In PCAP capture mode instead, packets will be captured from a PCAP file, which is specified by the user by passing the file global path in a proper configuration field (the file needs to be previously uploaded in the access point). This latter capture type is useful for reproducible research, using publicly available data sets usually stored in the form of PCAP files.
6) *Features List:* Feature-Sniffer allows the user to select the traffic features to be computed for each time window and each associated device. The most popular features used in network traffic analysis are available for extraction. Such features include several statistics regarding the packet size (72 features), the payload length (63 features), and the packet interarrival times (45 features). The available statistics include count (#), sum (Σ), mean (μ), median ($\mu_{1/2}$), mode (M), variance (σ^2), standard deviation (σ), and Kurtosis (K). Specific options for TCP and UDP packets can be selected, or packets carrying any of the two transport protocols can be considered in an aggregated fashion. The same holds for Downlink (DL) and Uplink (UL) packets. Additionally, the user can select 144 features derived from the probability mass function (PMF) of packet lengths, discretized in intervals of 100 bytes (i.e., containing the number of packets with length in range [0,100] bytes, [100,200], and so on ...), again with the possibility of differentiating between uplink/downlink and TCP/UDP packets. Furthermore, we include five more features containing the number of remote IP addresses contacted by a device in each time window, the number of local TCP/UDP used ports, and the number of remote TCP/UDP ports contacted. In total, the user can select up to 329 features extracted from the network and transport layers.
Finally, to complete the potential information to be extracted using the proposed tool, we extend the set of available features, including some physical layer characteristics, that can be computed when operating the tool in live capturing mode. In particular, the user can select statistical features based on receiver signal strength indicator (RSSI) and CSI from frames sniffed from the selected devices, specifying the statistical operation (e.g., mean and variance) and the interface collection parameters (e.g., Wi-Fi band and Wi-Fi channel). Many

[2]https://github.com/openwrt/luci

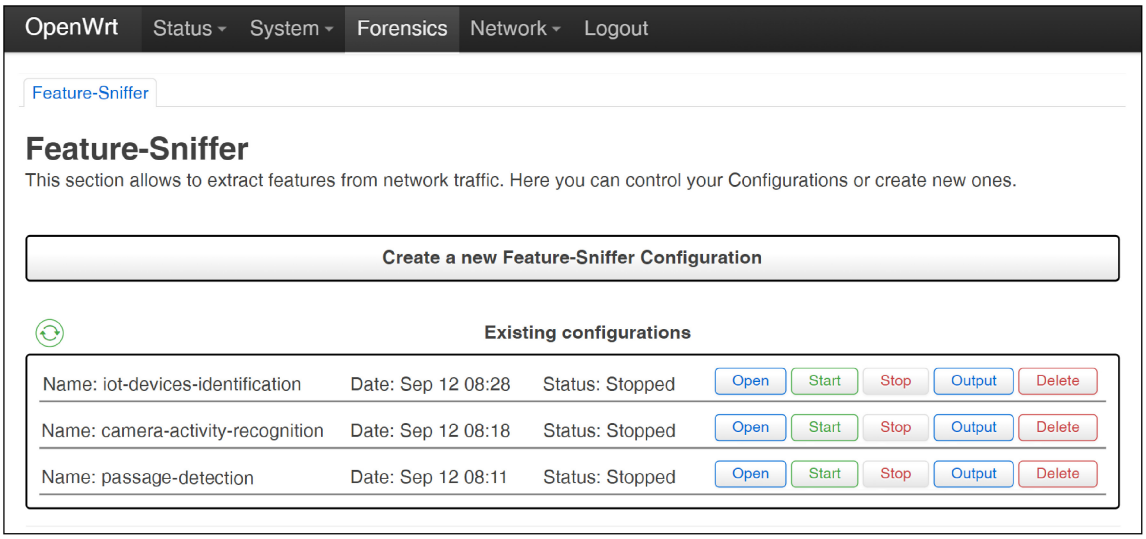

Fig. 2. Feature-Sniffer GUI homepage. The user can control existing configurations or create new ones for collecting network traffic features with custom parameters.

works in the literature have shown that such features are appropriate for unveiling potential information about the surrounding environment, such as detecting human movements. RSSI expresses the power of the transmission at the receiver, while CSI returns a complex value for each subcarrier of the OFDM channel, each representing the amplitude attenuation and phase shift of the transmission. The user can decide to extract statistical values for each subcarrier, i.e., mean amplitude of the frames in a time window for each subcarrier, or to aggregate over all the subcarriers using an additional statistical function to obtain a 1-D feature for each window for each device. The user can select a total of 20 combinations of features extracted from CSI data: four by selecting mean, standard deviation, variance, and Kurtosis for each subcarrier, and 16 by combining the four with a subsequent aggregation over all the subcarriers. For the RSSI value, instead, six statistical features can be selected for each time window: 1) mean; 2) median; 3) mode; 4) standard deviation; 5) variance; and 6) Kurtosis.

7) *Output Parameters:* Feature-Sniffer allows the creation of customized outputs, such as the addition of a ground-truth label for each device (useful, e.g., for device identification tasks), the use of feature headers name in the output, the possibility to create one file for each different associated device or the use of relative or absolute timestamps for each time window. The user can also decide to periodically switch output files specifying the switching period duration and the maximum number of files to generate, an option that could be used to periodically transfer files to external entities and hence extend the storage capabilities. This last functionality allows defining a closed-circuit behavior in which a predefined number of files are recycled indefinitely until the user decides to stop the process, avoiding the risk of filling the access point storage and hence unintentionally killing the collection process.

D. Feature Extraction Engine

Once a configuration is set up, it can be started by clicking on the start button in the user interface. This will trigger the transmission of an HTTP request to the Web server on the access point, which in turn will start the FEE according to the selected configuration parameters. In particular, the extraction process is divided into two different components: one for the collection of network/transport layer features (FEE) and another for the physical layer characteristics (P-FEE). The FEE process is a C program running inside the access point and working as follows: first, all configuration parameters are loaded with the help of the libconfig library[3] APIs. Then, two concurrent threads are started: one thread is in charge of capturing packets for each associated device, using the `libpcap` library [28], and grouping them in consecutive time windows according to the user-defined window length and packet/device filters. When a window is ready, the thread pushes it into the input queue of a second thread, which is in charge of computing the selected features on the packets contained in the window and saving the result to a comma separated value (CSV) file. Such a decoupled approach for capturing and processing network traffic is chosen to avoid losing packets due to the computational burden of feature extraction and the corresponding write operations to the output files.

Whenever the user wants to interrupt the packet capture, it can use the stop control button from the Feature-Sniffer interface to trigger a kill signal for the corresponding process. In particular, to avoid losing some of the windows in the output, we have redefined the handler of the `SIGINT` signal: the signal triggers the break of the capture loop of the main thread so that no additional packets are considered; all the not yet considered windows are then loaded into the output queue so that the secondary thread can process them and finally print them in the output file. Finally, all the output files are closed, and the FEE process can terminate. In the particular case of PCAP file capturing mode, the packet capture automatically

[3]https://github.com/hyperrealm/libconfig

terminates when the process has parsed all the packets in the selected file.

E. Physical Feature Extraction Engine

Differently from the previous case, the collection of CSI cannot be done from all network interface cards (NICs) and requires proper hardware and software. For this reason, after an accurate analysis of the existing solutions reasoning on the tradeoff between costs and processing capabilities, to build the P-FEE, we selected the Nexmon CSI tool [29] to be installed in a Raspberry Pi 3B+, connected to the access point using a standard Ethernet cable. Nexmon CSI is a simple software that allows the extraction of CSI characteristics from frames collected from a wireless interface set in monitor mode, after a proper selection of the collection parameters (interface, Wi-Fi band, channel bandwidth, and channel number). To control the Nexmon collection process with the parameter selection and the corresponding output management and integrate it with the Feature-Sniffer functionalities, we create an MQTT link between the access point and the Raspberry. The MQTT protocol perfectly matches our requirements, thanks to the publish-subscribe pattern and its lightweight characteristics. In particular, we set the Raspberry Pi to create an instance of the mosquitto [30] broker and to run a subscriber to receive commands incoming from the access point to control the Nexmon collection process. As the user selects CSI/RSSI features in a configuration and starts the Feature-Sniffer capture, the access point communicates the configuration parameters sending a publish message to the start topic in the broker so that the subscriber process can properly tune the Nexmon CSI and start the collection. With the same logic, to stop the collection of such features, as the user presses the stop button, the access point sends a publish message to the stop topic that will trigger the termination of the Nexmon process in the Raspberry Pi. For what concerns the RSSI/CSI data processing and statistical feature extraction, as the user requests for the output, a publish message directed to the download topic triggers the Raspberry Pi to process the data collected from the Nexmon process through a simple python script, to eventually produce a CSV file containing the selected RSSI/CSI features for each time window for each device.

F. Output Management

The Web interface allows the user to download the output either when the FEE process has finished or while it is under execution. By using the proper download button in the GUI, the user triggers a shell script to 1) request the physical layer CSV output through an MQTT subscription to the *output* topic and 2) prepare a compressed archive (using tar and gzip) with all the output files containing all the requested features produced from the FEE and P-FEE components. As the compressed output is ready, it is eventually sent back to the user as a response to the HTTP request, and it is ready for the analysis process.

IV. Performance Evaluation

A. Evaluation Setup

After the installation of the tool with all the required components, the access point becomes ready for the collection of potential information to be used in forensics processes. The network administrator should set at least one configuration to run in the access point and collect such information in order to be analyzed when needed. To understand the impact of the proposed tool in real-life smart home scenarios, we installed Feature-Sniffer on two commercial Wi-Fi access points supporting the OpenWrt firmware to present an evaluation of the computing performance. In particular, the following access points have been selected for our analysis.

1) *Access Point 1:* Linksys WRT3200ACM. This device is characterized by 512-MB RAM and 1.8-GHz CPU and has been selected for analyzing modern medium-sized networks.
2) *Access Point 2:* Netgear R6120. This device is equipped with 64-MB RAM and a less powerful 580-MHz CPU and has been selected to simulate more common small networks (i.e., with few IoT devices).

As a first step, we flushed the two selected access points with the most up-to-date compatible version of OpenWrt (version 21.02.0 for both), properly compiled for the two different system architectures. Once the operating system is up and running, we installed the Feature-Sniffer GUI as an extension of the LuCI Web interface, adding both the client (HTML/Javascript views) and server parts (Lua controller). Finally, to install the FEE, we compiled the C program for the two proper system architectures by using the corresponding OpenWrt Software Development Kits.

The compilation procedure produces a *.ipk* package to be installed into the system using the `Opkg` packet manager, a fork of `ipkg` used in OpenWrt-based systems. Finally, to extend the access point storage capabilities, we mounted a USB drive to the two access points to store the Feature-Sniffer output and easily move it to other machines for proper analysis.

To evaluate the computing performance, we set up a Wi-Fi network in our laboratory and let 10 IoT devices connect to each access point. The devices included in our experiments are reported with their traffic types in Table I. Each device is characterized by a different traffic profile: for what concerns IoT devices, we stimulate the production of traffic by interacting with them during the tests (e.g., turning on/off the smart bulbs or activating the smart cameras). Moreover, in order to test the performance of Feature-Sniffer under considerable traffic loads, we configured a Raspberry Pi 3B+ and a laptop as `iperf3` client and server, respectively. The `iperf3` tool allows the generation of different traffic profiles, controlling the traffic bandwidth, the packet size, the transport protocol, and the transmission duration. In order to tune the iperf3 tool parameters to be consistent with respect to modern IoT networks, we deeply analyzed a public IoT data set [8] containing network traffic traces from different common devices to understand the different traffic patterns of a vast variety of IoT devices and emulate them through

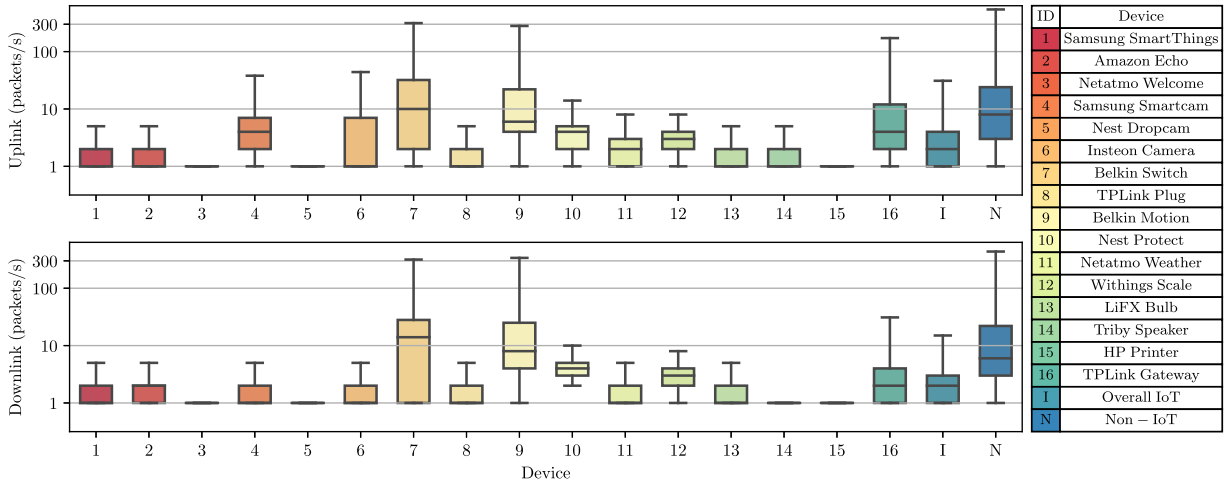

Fig. 3. Analysis of the traffic of devices in the data set in [8]. We report the box plot of the uplink (top) and downlink (bottom) rate, in terms of packets/s.

TABLE I
LIST OF DEVICES INCLUDED IN THE EVALUATION NETWORK

Device	Traffic type
Non-IoT devices	
Laptop ASUS Win 10	iperf3 server
Raspberry Pi 3B+	iperf3 client
IoT devices	
Tapo C100 Smart Camera	100Kbps video streaming
Teckin TC100 Smart Camera	100Kbps video streaming
Blink Indoor Smart Camera	Background
Tapo P100 Smart Plug	Control traffic from app
Ezviz T31 Smart Plug	Control traffic from app
Ledvance Wi-Fi Plug	Control traffic from app
Tapo L530E Smart Bulb	Control traffic from app
Netatmo Weather Station	Control traffic from app
Amazon Echo Dot 4	Traffic by speech control
Google NEST Protect	Control traffic from app

the iperf3 tool. In particular, we analyze the first 14 days of traffic traces involving 20 different devices: 16 IoT and four non-IoT. Fig. 3 reports the Downlink and Uplink traffic rate of the considered devices in the form of a box plot. For the computation of the plot, we consider only the active periods for each device. The two rightmost boxes in the figure report the cumulative IoT and Non-IoT traffic rates to understand the overall average traffic load of the network.

As a consequence of the data set analysis, we tune the iperf3 software to generate traffic with a maximum rate of 400 packets/s, given that this value is highly above the mean traffic produced from tens of IoT devices simultaneously connected. We keep track of the CPU and RAM consumption of the FEE process [tracked by process identifier (PID)] using two specific shell scripts, which are able to periodically measure the values with a sampling rate of two samples per second. For what concerns RAM usage, the monitoring script is programmed to read the memory usage from the FEE process as it is done from the `htop` tool,[4] a common software for real-time monitoring of resources usage in Linux-based systems. To extract the memory usage of the process, we follow the guideline reported in [31]: we take the value of the memory consumption from the `/proc/$PID/status/VmRSS/` file, which is updated from the operating system each time memory allocation changes for the process having `$PID` as unique PID. For the CPU usage, instead, we used the Linux `ps` command to track the percentage of CPU used from the FEE process, passing the correct PID as an argument.

[4]https://htop.dev/

B. Computational Performance

The performance evaluation phase is performed in the same way for the two selected access points under the same network conditions as explained above. We perform several tests with the goal of monitoring the performance of the Feature-Sniffer tool under different traffic loads while understanding its impact on the access points working operations. In particular, we created one Feature-Sniffer configuration, activating all available network/transport layer features to be extracted for each device in the network, setting the window size to 5 s. Features extracted from the physical layer are not considered here since they are computed in the Raspberry Pi and do not require additional access point resources.

1) Impact on Access Point Resources: To understand the difference in CPU and RAM enabling/disabling the Feature-Sniffer tool, the tests have been executed as follows: first, we start the Feature-Sniffer tool with only the IoT devices connected and run the monitoring script to keep track of the resource consumption. After 30 s, we start generating traffic triggering the various IoT devices and running the iperf3 client under different transmission rates. We generate the traffic for a total duration of 60 s, and at time $t = 90$ s, we stop the iperf3 tool and stop triggering IoT devices. Finally, we let another 210-s pass, and after 5 min of total execution, we stop the Feature-Sniffer process, together with the monitoring script. During the execution, the different IoT devices triggered produce, on average, a rate of 100 packets/s in total. To analyze the results under different transmission rates, we repeated the tests four times for each access point, each time changing the

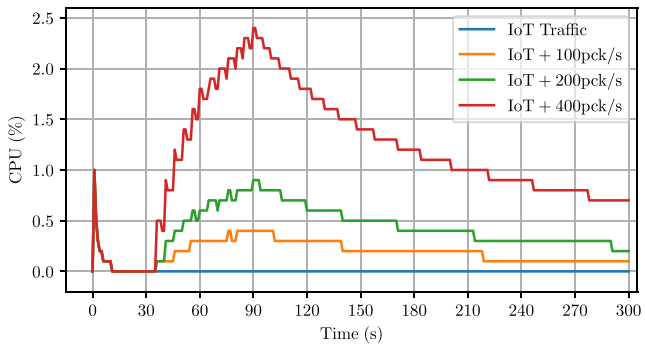

Fig. 4. Access Point 1 CPU utilization of the FEE process over time, using different network traffic loads (w = 5 s, all features).

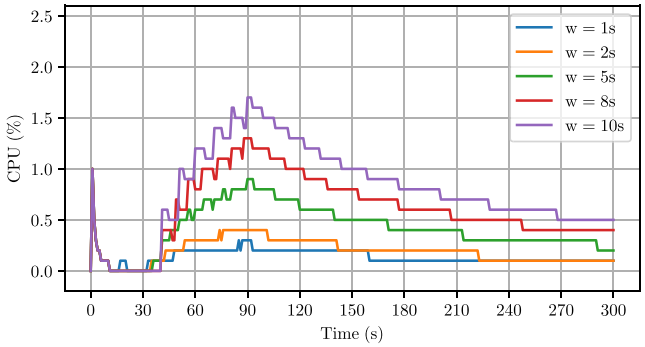

Fig. 6. Access point 1 CPU utilization of the FEE process over time, using different window duration values.

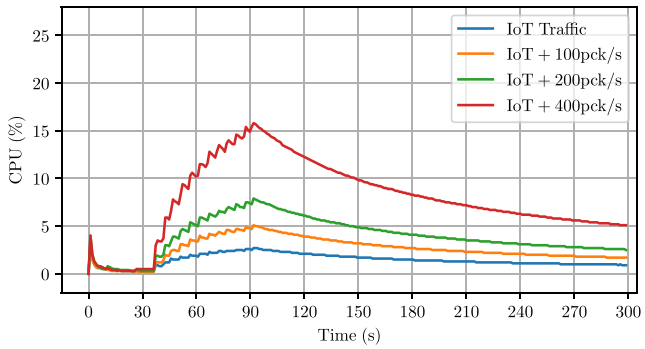

Fig. 5. Access point 2 CPU utilization of the FEE process over time, using different network traffic loads (w = 5 s, all features).

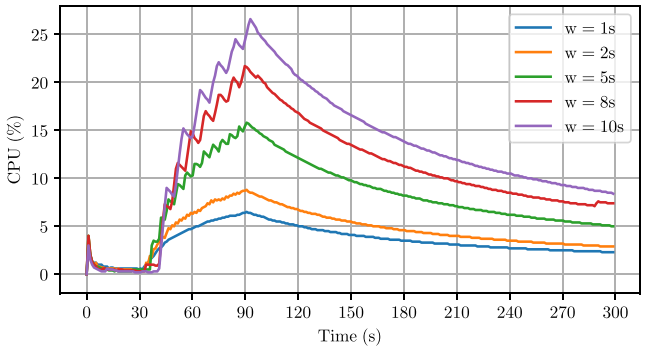

Fig. 7. Access point 2 CPU utilization of the FEE process over time, using different window duration values.

Iperf3 client traffic rate: 0 pcks/s (only the traffic produced from the connected IoT devices is considered), 100 pcks/s, 200 pcks/s, and 400 pcks/s. We report the CPU usage for the two access points in Figs. 4 and 5, respectively. As the two figures show, the CPU utilization remains low for the two access points, even if considering a rate of around 500 packets/s, equivalent to having around 50 IoT devices communicating simultaneously. In all analyzed cases for both access points, the average RAM utilization remains almost constant below 2 MB, regardless of the traffic rate. During the simulation, we also kept track of the packet error rate as reported by the iperf3 client-server interaction: no particular differences have been observed when executing/not executing the tool in the various considered cases.

As we reported in the previous figures, CPU utilization of the FEE process strongly reacts to the network traffic conditions. However, the reaction is not immediate, but we observe a shift of a few seconds depending on the window duration used in the Feature-Sniffer configuration. To analyze the CPU utilization under different time window values, we replicate the tests maintaining the same traffic rate and changing the window value used to aggregate packets to compute the features. Figs. 6 and 7 report the CPU utilization for the two access points when using a time window in the set [1,2,5,8,10] seconds to model the data set. Also in this case, we set a configuration with all the available features enabled and tune the iperf3 client to generate traffic at a rate of 200 packets/s, additionally to the traffic produced by IoT devices in the network. The two figures highlight that using a longer time window to model the data set leads to higher CPU consumption.

Finally, to investigate the behavior of the tool with higher traffic loads under different amounts of features selected, we also report the maximum CPU used by the FEE process in function of the transmission rates when using two Feature-Sniffer configurations: one with all the available features enabled and a lighter version representing the average use case, with only 1/3 of the available features turned on (selected randomly). As shown in Figs. 8 and 9, as the generated traffic rate increases, the FEE saturates the CPU in both access points. However, running the tool with a lower number of features maintains the execution affordable even at higher traffic rates, especially in Access Point 1.

2) Long-Term Analysis: As an additional task, we investigate the long-term behavior of the tool in real IoT networks and analyze the CPU required by the process under continuous activity from IoT devices. We try to answer the question if the tool can support a long period of execution and what are the impacts on the access point functionalities. Fig. 10 reports the CPU values over time under the same network environment detailed previously, using the two different access points. The long-term performance tests are executed for one full day. We set the Feature-Sniffer enabling all the features and using different window values to highlight the performance difference. For what concerns the network traffic load, in addition to the 10 physically connected IoT devices, we tune the iperf3 tool to

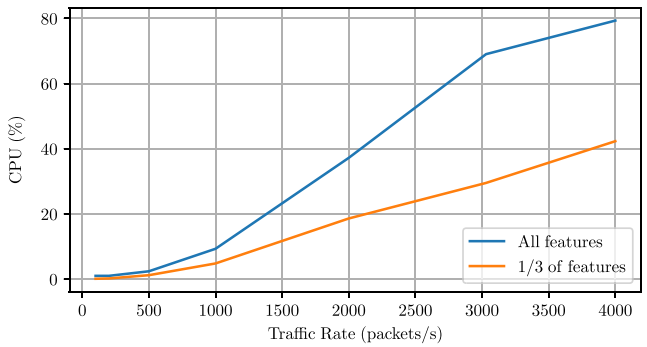

Fig. 8. Access point 1 maximum CPU utilization under different transmission rates. We set the FEE with all the features (blue line) and turning on only 1/3 of them (orange). The window value is set to 5 s.

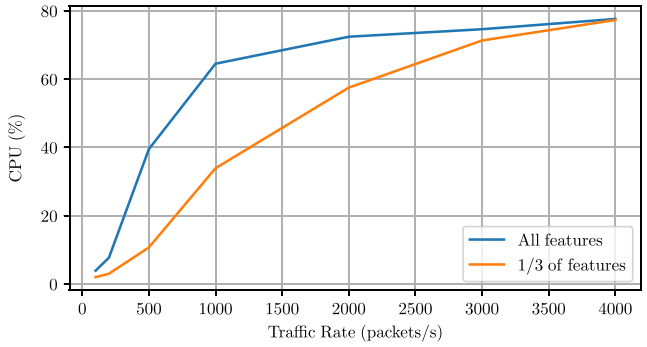

Fig. 9. Access point 2 maximum CPU utilization under different transmission rates. We set the FEE with two configurations, one with all the features (blue line) and the other enabling only 1/3 of them (orange). The window value is set to 5 s.

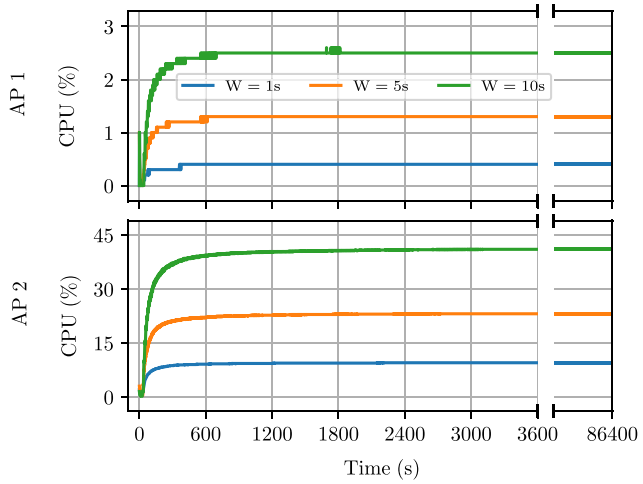

Fig. 10. FEE process CPU utilization (%) for Access Point 1 (top) and 2 (bottom) during a 1-day execution in a network with 30 active devices using different window lengths.

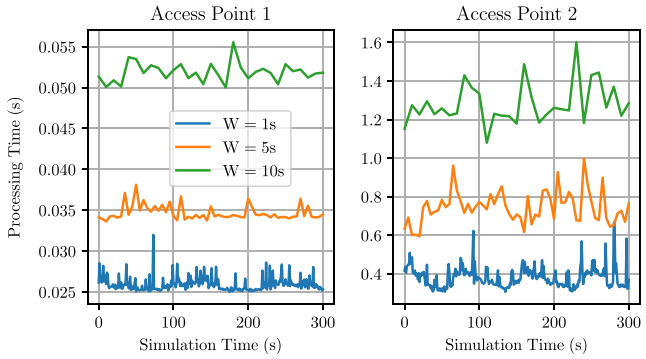

Fig. 11. Time required for processing the windows of 50 devices during the tool execution using different window lengths.

produce network traffic at a rate of 200 packets/s, thus simulating a network of around 30 continuously transmitting devices. As the reported figure shows, after around half an hour, the CPU required from the FEE process saturates to a fixed value and stops increasing, even if the traffic is still flowing and getting processed. This behavior confirms that for both access points, we can afford to run the tool for days without causing the system to overload.

3) Real-Time Analysis: To understand whether the tool can output features in a real-time fashion, we need to analyze the required processing time for each time window accurately. For this purpose, we added a new output entry to be printed along with the other selected features. Such entry computes the total time that the printed window requires in the processing phase: the processing time is computed considering the difference between the time in which the window is ready to be flushed by the computation thread and the time in which it is actually printed in the output file. For the system to be real-time, the sum of the processing times of each time window for all the connected devices should be lower than the window duration, otherwise, new windows would start filling the queue, causing the system to overload. For performing this simulation, we use the same testing environment as the previous cases. In this case, we keep the physical IoT devices connected and transmitting at their standard rate while we emulate 40 additional IoT devices to have an average behavior as stated in the previous analysis. Considering physical and emulated devices, we have a network traffic rate of around 500 packets/s belonging to 50 different MAC addresses. As for previous cases, we use different time window lengths since this value highly impacts the real-time behavior of the framework. Fig. 11 reports the total time required for processing the time windows of all the devices in a certain period of time for the two access points. The total processing time for a given time window is obtained by summing the processing times for all the devices in that time window. As reported in the two plots, the total processing time remains under the window length for both access points, thus confirming the real-time behavior of the tool. The results again confirm the clock speed difference between the two access points.

V. Use Cases

This section showcases the potential of the proposed solution in real-life IoT forensic investigations. For this reason, we define four different use cases in which the Feature-Sniffer output can be extremely useful for solving forensic processes: 1) IoT device identification; 2) human activity recognition;

3) human passage detection; and 4) voice assistant interaction classification. The first task we define involves a high range of IoT devices with different functionalities. To include as many devices as possible, we referred to a public data set (from [8]) containing data from 31 different popular devices and used it for the task of IoT device identification: the goal is to distinguish the IoT device only from its network traffic traces. This task can be the first step of an advanced analysis pipeline that includes subsequent device-specific tasks. For example, this task allows the exclusion of non-IoT devices from forensics analysis, avoiding storing information if the device is detected as non-IoT and hence reducing the load of the access point processing capabilities. Moreover, with a device identification configuration, it is possible to set proper Feature-Sniffer configurations to be enabled only if specific types of devices are detected, i.e., start a capture for speaker recognition with a proper predefined configuration if a home assistant is detected in the network. The second task we consider involves the network traffic extracted from consumer IoT video cameras. Such devices are particularly interesting from a forensic point of view, due to their widespread in smart homes/smart building scenarios: indeed, they constitute the perfect digital witness by design. However, for privacy and security reasons, IoT cameras usually stream encrypted video traffic directed to proprietary cloud services/mobile apps, making it almost impossible to retrieve the video content. Nevertheless, recent works from the state-of-the-art highlighted that some information leakages occur in encrypted video traffic, allowing for a partial understanding of the events happening in the FoV of the camera [13], [15], [16]. In a nutshell, variations in packet size and interarrival times could indicate activity in a video stream, regardless of the used video codec, camera, and processing hardware. The task that we aim to solve in this work using such devices is user Activity Recognition, with the goal of distinguishing the activity being performed by a human in front of the camera only using encrypted traffic traces.

To better understand the potential of Wi-Fi CSI, we include a third task to unveil information on the surrounding environment using physical layer features extracted from a generic IoT device: human passage detection. For the task, we use the CSI features extracted from a consumer smart home device to unveil the presence of a person passing through a room door. Finally, given the great popularity of smart speakers in smart homes, we conclude the application cases with a task that takes the goal of recognizing and classifying Amazon Echo user interactions, only referring to the traffic characteristics of the device.

In the experimental results presented in the next section, we show that Feature-Sniffer is a formidable tool to easily collect the traffic features needed to successfully accomplish the four tasks, at same time reducing the storage space needed for storing the corresponding potential evidence.

A. IoT Device Identification

The first task relies on a public data set containing traffic traces stored in PCAP files. The information extraction required to use the Feature-Sniffer tool in an offline capturing mode, directly processing the files instead of analyzing the live traffic from the access point interfaces. To reduce the classification task load, we use a subset of the public data set: the first 14 days of traffic from the available data are selected for the task. In order to ease the processing phase, after downloading the traffic traces using the HTTP APIs provided by the authors, we merged the 14 different files into a single PCAP file. For the task at hand, we set a configuration specifying the path of the single PCAP file, which is stored in a USB drive connected to the access point. Then, we select 140 total features: 32 from packet size (TCP DL, TCP UL, UDP DL, and UDP UL), 14 from Payload size (UL, DL), 25 from interarrival times (TCP DL, TCP UL, UDP DL, UDP UL, and Total), 64 from PMF features of packet length (TCP DL, TCP UL, UDP DL, and UDP UL), and five from local ports and remote IPs/ports contacted. We filled the MAC filter list with the physical addresses of the interested devices, to avoid parsing traffic from undesired entities. The experimental results are presented using a time window of 5 s: the full extraction process with the selected parameters produced as output a single CSV file having 20 labels (16 IoT and four non IoT) and roughly one million total entries.

B. Human Activity Recognition

For the second application case, we set up the IoT network using one of the two access points introduced in Section III-D and connect several IoT cameras to it. In particular, we selected the Linksys access point and three Wi-Fi IoT cameras of different brands having similar functionalities: Ezviz C6N, Tapo C100, and Teckin TC100. All cameras have the same resolution (1920×1080) and can be activated through a proper smartphone application. The three cameras have been mounted in an indoor space very close to each other, so that their FoV were overlapping. We configured each camera to stream video over Wi-Fi continuously, disabling the audio coming from the embedded microphones. Then, we selected four different activities to be performed in the FoV of the three cameras.

1) *No Activity:* A person sitting still in the FoV of the cameras.
2) *Slow Movement:* A person walking slowly and calmly in the FoV of the cameras
3) *Fast Movement:* A person frenetically moving in front of the cameras
4) *Lights On/Off:* The artificial light in the indoor space is turned on or off. No other light source is present.

Each activity has been performed for a period of 30 min and in order to produce one data set for each activity, we set up four configurations from the Feature-Sniffer GUI. Each configuration uses a 2-s window containing the following features, derived according to the study in [13] (58 features in total).

1) Downlink packet size (8 features: all statistical values).
2) Uplink packet size (8 features: all statistical values).
3) Downlink interarrival time (5 features: mean, median, variance, standard deviation, and Kurtosis).
4) Uplink interarrival time (5 features: mean, median, variance, standard deviation, and Kurtosis).
5) PMF features of downlink packet lengths (16 features: all size ranges).

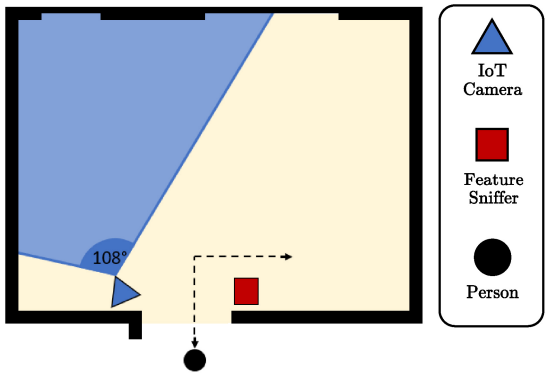

Fig. 12. Sketch of the disposition of devices in the room for the human passage detection task.

6) PMF features of uplink packet lengths (16 features: all size ranges).

To avoid the computation of features for unrelated devices, the device filter in the four configurations is filled with the MAC address of the three cameras. A different ground-truth label is used for each activity-camera pair: numbers from 1 to 3 for the No Activity configuration (Ezviz = 1, Tapo = 2, Teckin = 3), numbers from 4 to 6 for slow movement and so on for a total of 12 labels.

For each activity, we start the proper configuration to extract features while performing the activity in front of the cameras. The fourth activity (Lights) has been performed being sure that each 2-s time window observed exactly one light switch. After 30 min, we stop the configuration and download the obtained features for later analysis. The final data set therefore contains roughly 900 observation windows per activity per camera, each one containing 58 traffic features and one ground truth label.

C. Human Passage Detection

The third forensic task we present in this work takes as a goal the detection of the passage of a person through the room door, either entering or leaving the room, relying only on CSI measurements. Since the aim of the task is to detect the passage only using physical layer characteristics, the type of device is not strictly related to the task itself. For this reason, we selected a commercial IoT camera, considering that such devices are usually characterized by a high-packet generation rate. However, in our previous studies, we also considered the case in which lower transmission rates are used for similar tasks, and we achieved good results. Thus, we can assume that the analysis considered here can be generalized using other types of IoT devices with different network traffic patterns [32]. We place the smart camera and the access point with Feature-Sniffer on the two sides of the room door, as in Fig. 12, and we start the feature collection while the person moves in and out of the room walking in the dashed line, alternating the passage with idle periods in which the person is not moving. Differently from the second use case, during the data set collection, the FoV of the camera does not overlap with the area in which the person is moving, so a variation in the camera FoV does not produce a variation in the traffic rate. Moreover, to be sure that the camera is transmitting sufficient traffic, we enable video streaming from its smartphone application while collecting the data. For the data collection, relying on the methodology presented in [33], we create a new configuration in Feature-Sniffer and compute a single feature based on CSI values extracted from the frames of the IoT device. The selected feature aggregates the frames into time windows of fixed duration and extracts, for each window, the standard deviation of the CSI amplitude of the frames for each subcarrier. Then, for each time window k, the standard deviations over all the subcarriers are averaged to obtain the selected feature A^*, that is: $A_k^* = \mu([\sigma_k^i \ \forall i])$, with i ranging in the set of available subcarriers.

The device filter of the configuration is filled with the MAC address of the selected smart camera, and the window size is assigned to 3 s, which is the average duration of the walk of a person through the room door. Finally, the data set is constructed starting the configuration while performing the human activity several times. In total, we collected data performing 50 passages, 25 to get inside and 25 to get outside the room, taking note of the exact activity timestamps to assign the ground truth labels.

D. Smart Speaker Interaction Classification

The last application case we present in this work involves the analysis of the network traffic produced by an Amazon Echo voice assistant device, one the most common type of IoT device present in modern smart homes. Considering that such types of devices offer many services and act as home hubs for other IoT devices, they can reveal much potential information about the user, the environment, and the devices they allow to control. For this reason, we use Feature-Sniffer to extract network traffic features and use them for three different subtasks: 1) detecting the user interaction with the device; 2) distinguishing the user language; and 3) distinguishing the voice nature of the speaker, i.e., whether real or synthetic speech is being used to trigger the device. For the data set construction, we decided to split the traffic collection pipeline into two different steps: first, speech samples are collected from different human speakers reading different questions in two languages (Italian and English). To obtain homogeneous traffic, we balanced the age and gender of the speakers. In addition, to augment the set of audio samples used to trigger the device, we reproduce the same set of questions for both languages with synthetic voices using three different text-to-speech (TTS) APIs: Google TTS,[5] pyttsx3[6] (based on Python), and Amazon Polly.[7] In total, we obtained speech samples from 31 Real Speakers (22 Italian and nine English) and 23 Synthetic Speakers (15 Italian and eight English), each asking 20 questions. Furthermore, each question is repeated three times. The final data set thus contains a total of 3240 voice audio samples, each used to produce an interaction with the Amazon Echo device while the traffic is recorded. The network traffic is collected using the Feature-Sniffer tool, while the voice samples are reproduced using a Raspberry Pi 3B+

[5]https://cloud.google.com/text-to-speech
[6]https://pypi.org/project/pyttsx3/
[7]https://aws.amazon.com/polly/

to activate the Amazon Echo device, which was previously plugged and connected to the access point. While reproducing the speech samples, ground truth information, such as the interaction timestamp, language, voice nature, and question, being performed is collected. For the traffic collection, we selected a subset of traffic features and used a 1-s time window length, motivated by the generally short duration of the collected speech samples (in the order of 1–2 s). The selected features (25 in total) only consider the TCP protocol, which is the main used for the echo device communication, and refer to statistical values of packet sizes and interarrival times. The selected features mostly refer to Uplink traffic since it characterizes the most user interactions. As the data set is collected, we identify three different analysis tasks to be performed on top of the extracted features. The first goal (task 4a) is to distinguish whether the user is interacting with the device or not, i.e., the user is speaking to the device after pronouncing the activation word. For this purpose, we used the obtained data set and labeled all the time windows in which the device was receiving a question with label = 1, while we used label = 0 under silent or passive periods. After the interaction detection task, we define two different classification tasks that aim at categorizing the interactions. The first one (task 4b) takes the goal of distinguishing the language of the question between Italian and English, while the second one (task 4c) aims at identifying a synthetic voice against a real person's voice. For these tasks, all the time windows containing an interaction have been labeled with the proper language (Ita/Eng) and with the proper voice nature (Real/Synthetic). Silent windows are not considered for the latter two subtasks to keep the classification with only two labels (binary classifiers).

VI. Experimental Results

This section reports the experimental results obtained by applying machine learning algorithms to the data obtained from the configurations explained in the previous section for the three different tasks. In particular, we have downloaded the features extracted from Feature-Sniffer on a laptop (Intel i7-8750H with six CPUs @ 2.2 GHz, 16 GB of RAM) running Ubuntu 20.04.4 and Python Version 3.7 with the use of the scikit-learn library[8] to apply several algorithms for the three different use cases.

A. IoT Device Identification

To perform IoT device identification using the data set previously described, after extracting the output from the tool we have applied several supervised machine learning algorithms for multilabel classification (random forest, adaptive boosting, *K*-nearest neighbor, artificial neural network, and Naive Bayes). In this first application case, the data set from the tool is ready for the training since the labels are already assigned from Feature-Sniffer. However, to reduce misclassification between non-IoT devices, we decided to merge the features from the two smartphones and two laptops into a single class assigning them the same label = "Non-IoT." In order to perform the training-test split, we applied a time-ordered sevenfold cross-validation approach: the 14 days data set is split into seven nonoverlapping folds, each made of two consecutive days of traffic features. Then, seven iterations are repeated in which sixfold are used for training and the remaining one for testing. We choose this approach to simulate a real use case, in which the train and the test sets do not contain features relative to the same time periods (e.g., train today and test tomorrow).

We achieve the most promising classification results using the random forest, *K*-nearest neighbors, and artificial neural network classifiers. The *K*-nearest neighbors has the advantage of a very limited "training" time at the cost of the testing phase, in which the algorithm computes all the distances with the other entries in the data set to produce the classification outcome. Considering the consistent size of the data set, we focus here only on the random forest classifier (whose results are reported and discussed here), in which there is a consistent training phase, performed once, at the advantage of requiring a limited time for future evaluations. The confusion matrix obtained using this latter algorithm is reported in Fig. 13, showing that the results are very promising, with an overall classification accuracy and F1-score of 0.94. As it can be seen from the matrix, almost all the devices are classified with over 90% accuracy. The main classification errors occurred between two devices of the same brand (Belkin Wemo Switch and Belkin Wemo Motion Sensor), achieving a balanced accuracy of 87% and 78%, respectively.

For completeness, we report here the results using the other classifiers. The *K*-nearest neighbors achieved a balanced accuracy of 88.7%, while for the neural networks, we tested four different models having 1 or 2 hidden layers and 140 or 280 different neurons per layer. The most performing neural network has two hidden layers and 140 neurons, reaching an accuracy and F1-Score of around 92.8%. However, as the numbers suggest, the best results are still achieved using the random forest classifier, which also holds for tasks 2 and 4 in this work.

B. Human Activity Recognition

For the second task of human activity recognition, we have trained different machine-learning classifiers for supervised multilabel classification with the goal of classifying the activity being performed in front of the smart cameras. For the task at hand, first we have merged the four outputs in one single CSV file, being this possible since the labels have been assigned properly to avoid conflicts. Then, for the activity recognition phase, we identified two approaches.

1) A one-shot classification, which can infer the activity from the network traffic, regardless of the camera producing it.
2) A two-way approach, based on a first identification of the camera producing the traffic, using a logic very similar to the first task, and a successive activity recognition using a different pretrained classifier for each camera.

The first method required us to merge all the labels referring to an activity to a single common label regardless of the

[8]https://scikit-learn.org/stable/

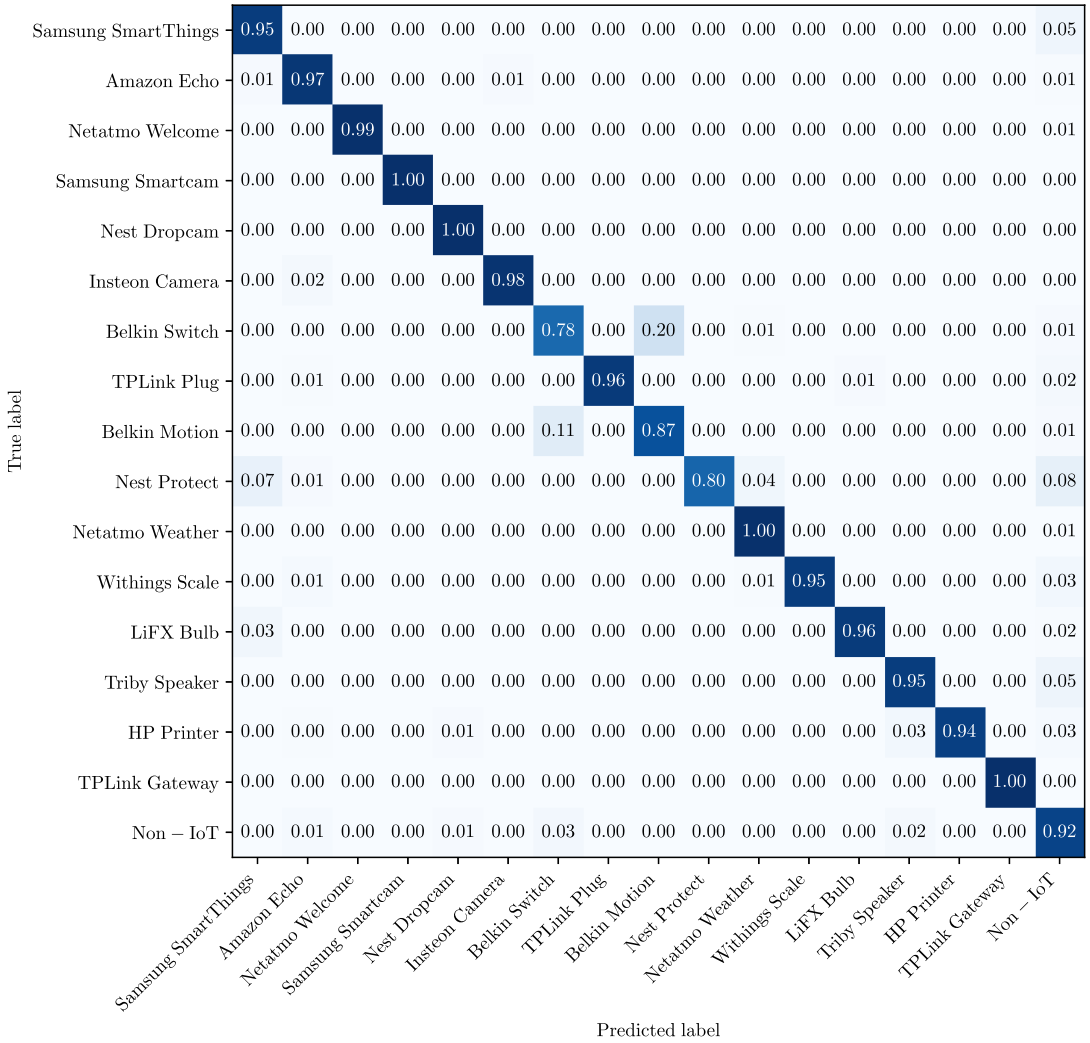

Fig. 13. Random forest classifier confusion matrix for the IoT device identification task using a sevenfold cross-validation approach on devices from the data set in [8]. The last class contains the traffic from four non-IoT devices (two smartphones and two laptops).

traffic source: we renamed labels (1, 2, 3) for No Activity as label = 1, labels (4, 5, 6) for slow movement as label = 2, labels (7, 8, 9) for fast movement as label = 3 and finally, labels (10, 11, 12) for lights On/Off as label = 4. We relied on a tenfold cross-validation process: the data set is split into tenfold and at each iteration nine of them are used for training and one for testing. We trained different classifiers and reported here only the best performance, obtained using the random forest. Fig. 14(a) shows the confusion matrix obtained in this case: as one can see the performance results are good, with an average F1-score of 83% and the majority of classification errors occurring between the activities of slow and fast movement.

In the second method, we try to improve the classification performance using a camera identification classifier to specialize the activity recognition training procedure. For the first step, the labels (1,4,7,10) in the merged data set have been renamed as Ezviz, while the labels (2, 5, 8, 11) and (3, 6, 9, 12) as Tapo and Teckin, respectively. The random forest classifier achieved again the best results with camera identification accuracy, precision, and recall of 99.88%. As the camera is identified correctly, the process follows with an independent activity recognition classifier for each camera, resulting in three different classifiers that need to be trained for this step. The resulting confusion matrices for the activity recognition, considering previous knowledge of the camera involved, are reported in Fig. 14(b)–(d). As observed in the figures, the best results have been achieved by using the Tapo camera, with an average accuracy of 96.6%, while the worst results are observed for the Ezviz camera, with an accuracy of 65%. The Teckin camera has obtained good results with an overall activity recognition accuracy of 90%.

C. Human Passage Detection

As already introduced in the previous section, the third task for human passage detection is performed by collecting a single feature extracted from CSI data produced from the smart camera for each time window. The first step is the construction of the ground truth based on the timestamps of the activities performed: we assign a label equal to 1 if the window is characterized by the person passing through the door, and 0 otherwise. As the data set is labeled, we design a binary

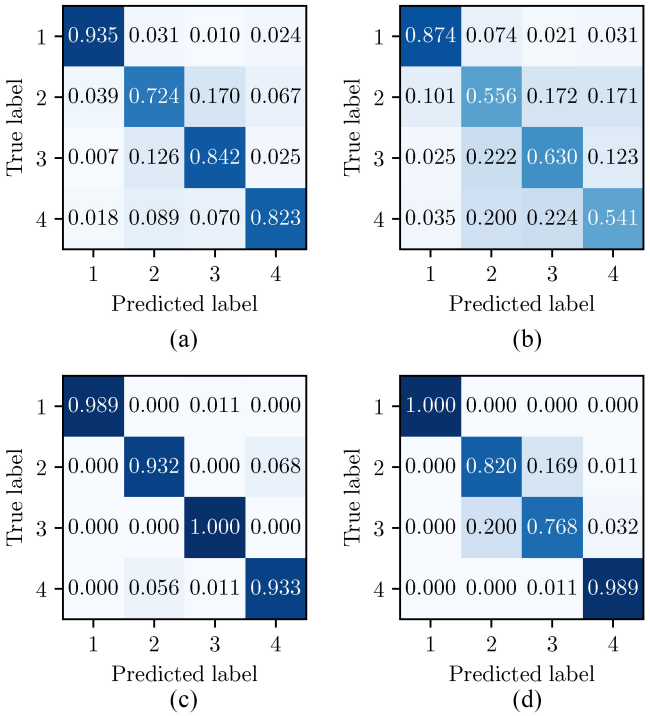

Fig. 14. (a) Random forest confusion matrix for activity recognition using one classifier for the three cameras. (b)–(d) Training a specific classifier per camera.

classifier that takes as input the single feature A^* and outputs a binary label Y as a prediction. We opt for a simple threshold classifier, according to the following:

$$Y = \begin{cases} 1, & A^* \geq \tau \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The choice of the proper value for the threshold τ greatly impacts the obtained performance and requires particular attention. For this reason, we perform several experiments with 1000 linearly spaced different values of τ, ranging between the minimum and maximum values assumed by the feature A^* in the data set. At each iteration, we compute the value of the prediction Y for each time window and we compare it with the corresponding ground-truth label, obtaining the number of true positives (TPs), true negatives (TNs), false positives (FPs), and false negatives (FNs). From such values, we compute the TP rate (TPR) and FP rate (FPR) for each threshold value. By representing TPR as a function of the FPR we obtain the receiver operating characteristic (ROC) curve, a well-known performance indicator used for binary classification. Moreover, to have an overall performance indicator, we also compute the area under the ROC curve (AUC). The ROC obtained for the classification is reported in Fig. 15, showing very good results with an AUC of 0.975.

D. Smart Speaker Interaction Detection

As previously stated, the fourth and last analysis task presented in this work distinguishes three different subgoals that involve the use of the Amazon Echo device. The first forensically relevant information to be extracted from a voice assistant device is to recognize whether the user is interacting or not with the device, i.e., a person activated the device asking a question. For this goal, as the labeled data set has been

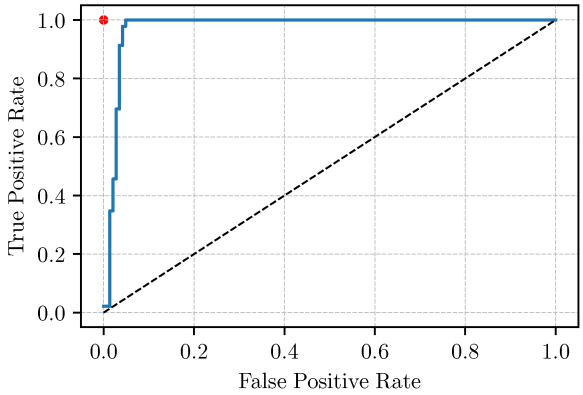

Fig. 15. Task 3: Human passage classification ROC curve. The AUC is 0.975.

constructed, we trained a binary random forest classifier using a tenfold cross-validation approach with the 25 TCP-based features extracted using the Feature-Sniffer tool. The interaction classification task shows almost perfect results with an over 99% F1-Score, as shown in Fig. 16 where the ROC curve is reported. Such excellent results are obtained thanks to the dependence of the uplink traffic on the state of the device. Indeed, uplink traffic peaks are observed whenever a person interrogates the device, as speech samples are transmitted to Amazon Web Services servers for processing.

Once an interaction has been detected by the proposed algorithm, we define two different subtasks to distinguish the language of the interaction (Italian versus English) and the nature of the speaker interacting with the device (real versus synthetic voice). For these tasks, since the same interaction can consist of more time windows depending on the question duration, we aggregated the statistics of more time windows belonging to the same interaction in one unique entry to be used in the data set: in total, 30 different features are computed for each interaction, obtained aggregating the previously computed 25 values of different windows with simple statistical functions (sum, mean, standard deviation). After this processing step, each interaction in the data set consists of one entry having 30 different features and a label based on the language of the speaker for task 4b and on the voice nature for task 4c. The obtained labeled data sets are then used to train one binary classifier per task. Again the selected approach for the two tasks is a tenfold cross-validation using random forest. The resulting ROC curves for these latter two defined subtasks using 1-s time windows for extracting the features are reported in Fig. 16. As shown, the two tasks have lower scores than interaction detection: around 86% for language recognition and 76% for voice nature recognition. The language recognition results, even if not perfect, are still promising, considering that we are only referring to encrypted traffic characteristics. For the latter synthetic voice recognition task, instead, the relatively bad results can be explained by the evolution of the TTS APIs, always producing more realistic voices. The Amazon Polly tool with neural optimization, indeed, produces a very realistic voice that can be easily confused with a human voice.

E. Dimensioning Window Length

After presenting the results for the four different tasks using a fixed time window value to aggregate packets and compute

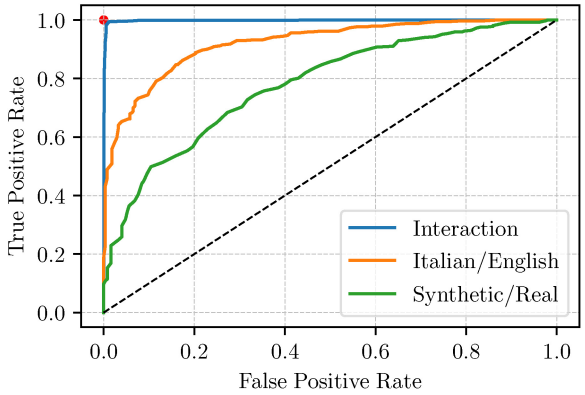

Fig. 16. Amazon Echo interaction classification ROC curve for the three different defined subtasks.

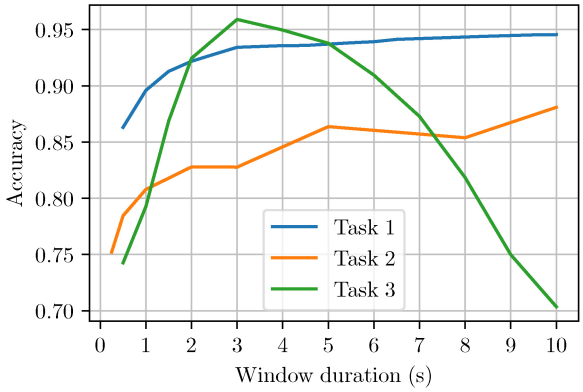

Fig. 17. Classification accuracy for the first three tasks using different window lengths for modeling the features.

the traffic statistical features, we propose an accurate analysis of the impact of the window length on the classification results. For this reason, we repeated the tests using different time window lengths in the range [0.5,10] s, and we proceeded with the analysis as discussed in the previous sections. In Fig. 17, we report the results obtained using different window lengths for the first three tasks analyzed in this work. The figure shows that different window lengths lead to different performance results during classification. For the first two tasks, the performance increases with larger windows and reaches its maximum when using 10-s windows. For the third task, instead, the classification performance hits the maximum when using 3-s time windows, while it is lower when using shorter or larger window lengths. The motivation for this behavior is that the passage of a person through the room door, on average, takes around 3 s. Hence, using 3-s time windows allows to isolate in a single window the full activity. When using bigger time windows, the aggregated values are balanced by the periods in which the person is not moving, thus making the value less dependent on the activity. For what concerns instead the fourth task based on the Amazon Echo device, we analyzed the classification performance using time windows of 0.5 and 1 s, considering that most of the questions activating the device in our data set have a duration lower than 2 s. The results, expressed in terms of random forest classification F1-Score, are reported in Table II. As shown in the table, the classification performance is not impacted by the window length.

TABLE II
TASK 4 CLASSIFICATION F1-SCORE USING 1-S AND 0.5-S WINDOWS FOR MODELING THE DATA SET FEATURES

Task	1-second F1	0.5-second F1
Interaction Detection	0.991	0.989
Ita vs. Eng	0.864	0.865
Real vs. Synthetic	0.766	0.758

TABLE III
PCAP AND CSV DATA SET SIZE BEFORE/AFTER GZIP COMPRESSION FOR THE FOUR DIFFERENT TASKS

	Task 1	Task 2	Task 3	Task 4
PCAP	7.98 GB	2.73 GB	20.5 MB	473 MB
PCAP + gzip	3.55 GB	2.16 GB	11.6 MB	343 MB
CSV	587 MB	3.02 MB	1.23 MB	70.3 MB
CSV + gzip	56.8 MB	865 KB	590 KB	7.43 MB

F. Required Storage

1) Lossless Compression: To highlight the benefits of using Feature-Sniffer in terms of used storage, we captured the complete network traffic traces produced for the different tasks also using *tcpdump* to store them in standard PCAP files. For each of the three tasks we perform a storage analysis comparing the data set size when storing the PCAP file containing the full traffic traces, and the CSV file produced from the tool, containing only the required features. In addition, we apply the gzip lossless compression algorithm to both the PCAP and the CSV for a further reduction of the data set size. The obtained data set sizes for the three tasks are reported in Table III. As the numerical results report, by compressing the CSV one can obtain a consistently higher compression ratio with respect to compressing the PCAP file. Results show that using our tool one can perform IoT forensic analysis with a very reduced data set, requiring up to $2,5 \times 10^3$ times less storage space (in task 2) than traditional approaches based on PCAP files.

2) Lossy Compression: To complete our discussion on storage occupation, we present an advanced analysis for its possible optimization. After proposing the results obtained by using the gzip algorithm for lossless compression, we opt for lossy compression techniques to encode each entry in the CSV file and further reduce the obtained data set size. Lossy compression algorithms can greatly reduce the data set occupation with high-compression ratios, at the cost of introducing noise in the compressed samples. Using such algorithms allows us to decide the number of bits to allocate to each data set entry and hence to control the final data set size, with a corresponding data reconstruction error strictly dependent on the compression ratio. The reconstruction error usually leads to a loss in the machine learning classification performance and thus there is a tradeoff between the storage occupied by the compressed data set and the obtained task accuracy. In order to study the storage-accuracy relation using lossy compression techniques, we selected the scalar quantization algorithm to encode the entries of the data set in the three different tasks and trained the machine learning models with the obtained compressed data sets. In particular, the scalar quantization approach first normalizes the entries in the data set and then encodes each value

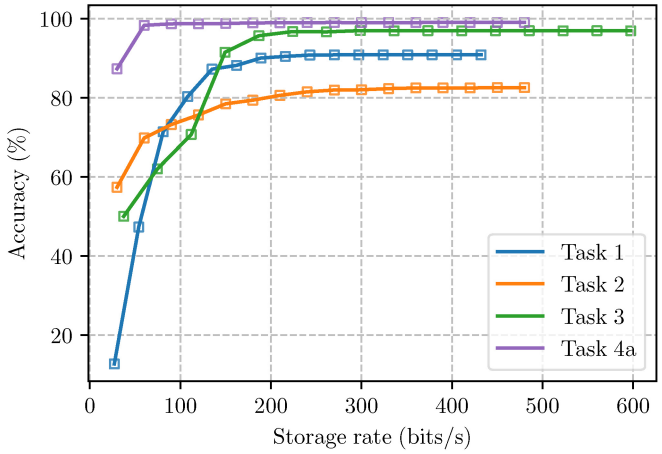

Fig. 18. Classification performance for the four IoT forensic tasks in function of the storage rate required to maintain the data set, using scalar quantization to compress the entries.

as an integer in the range $(0,2^b-1)$, thus allowing representing it with b bits.

We decided to compress the full set of features involved for the four tasks using different numbers of bits in the range [1,16], and for each obtained data set, we proceed with the analysis pipeline with the reconstructed data, training the classifiers and extracting the classification performance. In particular, for the first task, we compressed the selected 140 features collected for each 5-s window; for the second task, we compress the 58 different 2-s windowed features. For what concerns the third task, we compress the standard deviation of the amplitude and phase in each subcarrier (56 nonnull subcarriers) for the 3-s windows. In this case, the second aggregation to obtain the single feature required for the classification task is performed after the data reconstruction step. Finally, for the last task, we compress the 25 TCP-based features for each 1-s time window.

Clearly, the different number of features used in the tasks and the different window values used for aggregation lead to different storage requirements. However, the required storage over time for the four different tasks results to be in the same order of magnitude, as reported in Fig. 18. The figure reports the average machine learning classification performance obtained in the four tasks, in function of the data storage expressed in bits per second. Each line corresponds to a different task, and each point in the line corresponds to a different number of bits used to encode the corresponding data set entries. The reported performance metric depends on the task: we extracted the balanced accuracy for the first two tasks, while for the third task, we selected the AUC. For the last task, we report the binary classification F1-Score. As the results show, the classification metric in the four tasks reaches the optimal value using only around 200 bits of storage every second. The results are good considering that under this rate, the daily storage occupation is around 2 MB per device (in the worst case, considering the device is continuously transmitting) against the average PCAP file size that may reach the order of GBs per day, depending on the device. In our previous work presented in [34], we deeply analyzed the storage-accuracy tradeoff for the first two tasks presented in this work, proposing a framework to extract the best operational point, expressed as the combination (F, B) of the number of features F and the number of bits B to use, for the optimization of such tradeoff.

VII. Conclusion

This work has presented Feature-Sniffer, a tool to turn an OpenWrt-based access point into a forensic device capable of collecting IoT device traffic features to be used as potential sources of evidence in forensic processes. After presenting the tool architecture and its implementation details, we perform a performance evaluation of the tool in modern Wi-Fi access points to analyze its impact in terms of CPU. In particular, we show that even small networks equipped with low-cost access points can use the proposed framework for performing network feature collection with a friendly graphical interface. Finally, we showcase potential uses in real-life forensic scenarios in which the traffic features extracted from consumer IoT devices can be used to unveil information from the surrounding environment. We defined four different tasks with the goal of 1) distinguishing the different IoT devices from traffic traces; 2) identifying the human activity performed in front of smart cameras; 3) detecting the human passage through the door with physical layer features; and 4) classifying user interactions with a smart speaker.

The experimental results are very promising and highlight that it is extremely simple to use Feature-Sniffer to set up a forensic gateway that allows the extraction of useful information from the devices and the environment in which they are located. To conclude the work, we provide an analysis of possible storage optimization techniques, showing the potential storage gain when applying lossy compression techniques to the output data set for the three tasks. Results show that 2 MB per day for each device is enough for unveiling the information for each of the four proposed tasks against the hundreds of MBs required for storing modern IoT network traffic traces in the form of PCAP files.

As future research directions, we plan to focus our research on further optimization of the feature storage using different and more complex compression schemes. In addition, we plan to expand the work currently done by enabling Feature-Sniffer with inference capabilities, bringing the inference procedure to the access point level and hence detecting real-life events from the network traffic features as soon as they occur, allowing specific behaviors if events are detected (i.e., store the full traffic and notify the user).

Acknowledgment

This study was carried out within the MICS (made in Italy-Circular and Sustainable) Extended Partnership.

References

[1] M. Stoyanova, Y. Nikoloudakis, S. Panagiotakis, E. Pallis, and E. K. Markakis, "A survey on the Internet of Things (IoT) forensics: Challenges, approaches, and open issues," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1191–1221, 2nd Quart., 2020.

[2] F. Palmese, A. E. C. Redondi, and M. Cesana, "Feature-sniffer: Enabling IoT forensics in OpenWrt based Wi-Fi access points," in *Proc. IEEE 8th World Forum Internet Things (WF-IoT)*, 2022, pp. 1–6.
[3] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. Internet Meas. Conf.*, 2019, pp. 267–279.
[4] Y. Wan, K. Xu, F. Wang, and G. Xue, "IoTAthena: Unveiling IoT device activities from network traffic," *IEEE Trans. Wireless Commun.*, vol. 21, no. 1, pp. 651–664, Jan. 2021.
[5] A. Acar et al., "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proc. 13th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2020, pp. 207–218.
[6] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "PingPong: Packet-level signatures for smart home device events," 2019, *arXiv:1907.11797*.
[7] A. J. Pinheiro, J. de M. Bezerra, C. A. Burgardt, and D. R. Campelo, "Identifying IoT devices and events based on packet length from encrypted traffic," *Comput. Commun.*, vol. 144, pp. 8–17, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419300052
[8] A. Sivanathan et al., "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1745–1759, Aug. 2019.
[9] Y. Meidan et al., "ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis," in *Proc. Symp. Appl. Comput.*, 2017, pp. 506–509.
[10] A. Aksoy and M. H. Gunes, "Automated IoT device identification using network traffic," in *Proc. ICC 2019- IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–7.
[11] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "IoT devices recognition through network traffic analysis," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, 2018, pp. 5187–5192.
[12] S. Li, K.-K. R. Choo, Q. Sun, W. J. Buchanan, and J. Cao, "IoT forensics: Amazon echo as a use case," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6487–6497, Aug. 2019.
[13] J. Wang, Z. Cao, C. Kang, and G. Xiong, "User behavior classification in encrypted cloud camera traffic," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.
[14] Y. Wan, K. Xu, F. Wang, and G. Xue, "IoTMosaic: Inferring user activities from IoT network traffic in smart homes," in *Proc. IEEE INFOCOM 2022-IEEE Conf. Comput. Commun.*, 2022, pp. 370–379.
[15] C. Wampler, S. Uluagac, and R. Beyah, "Information leakage in encrypted ip video traffic," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2015, pp. 1–7.
[16] H. Li, Y. He, L. Sun, X. Cheng, and J. Yu, "Side-channel information leakage of encrypted video stream in video surveillance systems," in *Proc. IEEE INFOCOM 2016- 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[17] J. C. Soto, I. Galdino, E. Caballero, V. Ferreira, D. Muchaluat-Saade, and C. Albuquerque, "A survey on vital signs monitoring based on Wi-Fi CSI data," *Comput. Commun.*, vol. 195, pp. 99–110, Nov. 2022.
[18] D. Caputo, L. Verderame, A. Merlo, A. Ranieri, and L. Caviglione, "Are you (Google) home? detecting users' presence through traffic analysis of smart speakers," in *Proc. ITASEC*, 2020, pp. 105–118.
[19] B. Charyyev and M. H. Gunes, "Misactivation detection and user identification in smart home speakers using traffic flow features," in *Proc. 14th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2021, pp. 135–146.
[20] M. Ford and W. Palmer, "Alexa, are you listening to me? An analysis of Alexa voice service network traffic," *Pers. Ubiquitous Comput.*, vol. 23, no. 1, pp. 67–79, 2019.
[21] C. Wang et al., "Fingerprinting encrypted voice traffic on smart speakers with deep learning," in *Proc. 13th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2020, pp. 254–265.
[22] J. Hyland, C. Schneggenburger, N. Lim, J. Ruud, N. Mathews, and M. Wright, "What a SHAME: Smart assistant voice command fingerprinting utilizing deep learning," in *Proc. 20th Workshop Privacy Electron. Soc., ser. WPES '21*, New York, NY, USA, 2021, pp. 237–243.
[23] S. Kennedy, H. Li, C. Wang, H. Liu, B. Wang, and W. Sun, "I can hear your Alexa: Voice command fingerprinting on smart home speakers," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Aug. 2019, pp. 232–240.
[24] T. Kalin, K. Stone, and T. Camp, "AMAZE: Recognizing speakers with Amazon's echo dot device," in *Proc. IEEE 16th Int. Conf. Mobile Ad Hoc Sensor Syst. (MASS)*, 2019, pp. 494–503.
[25] C. Meffert, D. Clark, I. Baggili, and F. Breitinger, "Forensic state acquisition from Internet of Things (FSAIoT): A general framework and practical approach for IoT forensics through IoT device state acquisition," in *Proc. 12th Int. Conf. Availability, Rel. Secur.*, 2017, pp. 1–11.
[26] L. Babun, A. K. Sikder, A. Acar, and A. S. Uluagac, "IoTDots: A digital forensics framework for smart environments," 2018, *arXiv:1809.00745*.
[27] Z. Liu, Y. Chen, W. Yu, and X. Fu, "Generic network forensic data acquisition from household and small business wireless routers," in *Proc. IEEE Int. Symp. "A World Wireless, Mobile Multimedia Netw." (WoWMoM)*, 2010, pp. 1–6.
[28] The TCPDUMP Group. "Libpcap." 2008. [Online]. Available: https://github.com/the-tcpdump-group/libpcap
[29] F. Gringoli, M. Schulz, J. Link, and M. Hollick, "Free your CSI: A channel state information extraction platform for modern Wi-Fi chipsets," in *Proc. 13th Int. Workshop Wireless Netw. Testbeds, Exp. Eval. Characterization*, 2019, pp. 21–28.
[30] R. Light, "Mosquitto: Server and client implementation of the MQTT protocol," *J. Open Source Softw.*, vol. 2, no. 13, p. 265, 2017.
[31] P. Ņikiforovs. "Htop explained: Explanation of everything you can see in htop/top on Linux." 2019. [Online]. Available: https://peteris.rocks/blog/htop/
[32] F. Palmese and A. E. C. Redondi, "Collecting channel state information in Wi-Fi access points for IoT forensics," 2023, *arXiv:2305.10554*.
[33] S. M. Hernandez and E. Bulut, "Adversarial occupancy monitoring using one-sided through-wall WiFi sensing," in *Proc. ICC 2021-IEEE Int. Conf. Commun.*, 2021, pp. 1–6.
[34] F. Palmese and A. E. C. Redondi, "A framework for storage-accuracy optimization of IoT forensic analysis," in *Proc. GLOBECOM 2022-IEEE Global Commun. Conf.*, 2022, pp. 3779–3784.

Fabio Palmese (Graduate Student Member, IEEE) received the M.S. degree in computer science and engineering from the Politecnico di Milano, Milan, Italy, in 2020, where he is currently pursuing the Ph.D. degree in telecommunications with the Advanced Network Technologies Laboratory, Department of Electronics, Information and Bioengineering.

His research activities focus on network traffic analysis and Internet of Things topics. His Ph.D. research project focuses on the IoT forensics, with specific attention to IoT network traffic collection and analysis for forensic investigations.

Alessandro Enrico Cesare Redondi (Senior Member, IEEE) received the M.S. degree in computer engineering and the Ph.D. degree in information engineering from the Politecnico di Milano, Milan, Italy, in July 2009 and February 2014, respectively.

He is currently an Associate Professor with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. From September 2012 to April 2013, he was a visiting student with the Department of Electronic and Electrical Engineering, University College of London, London, U.K. His research activities are focused on the design and optimization of IoT systems and on network data analytics.

Matteo Cesana (Senior Member, IEEE) received the M.S. degree in telecommunications engineering and the Ph.D. degree in information engineering from the Politecnico di Milano, Milan, Italy, in July 2000 and September 2004, respectively.

He is currently a Full Professor with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. From September 2002 to March 2003, he was a Visiting Researcher with the Computer Science Department, University of California at Los Angeles, Los Angeles, CA, USA. His research activities are in the field of design, optimization, and performance evaluation of wireless networks with a specific focus on communication technologies for the Internet of Things and future generation cellular networks.

Dr. Cesana is an Associate Editor of the *Ad Hoc Networks* (Elsevier).