IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 19, 2020, accepted July 16, 2020, date of publication July 27, 2020, date of current version August 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3012203

Packet Length Spectral Analysis for IoT Flow Classification Using Ensemble Learning

GENNARO CIRILLO AND ROBERTO PASSERONE, (Member, IEEE)
Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, 38123 Trento, Italy
Corresponding author: Roberto Passerone (roberto.passerone@unitn.it)

ABSTRACT With the proliferation of ubiquitous and autonomous devices for sensing, control, monitoring and conditioning, the Internet of Things (IoT) holds a great potential for the development of innovative applications. At the same time, network operators must support these devices with differentiated services, which rely on the ability to automatically recognize and classify the nature of the communication flows. In this paper, we present a supervised learning approach to discriminate between IoT and non-IoT traffic, and to determine the class of the device originating the packet flows. We make use of a reduced set of features based on the spectral analysis of the packet lengths of a flow, and evaluate an ensemble learning algorithm that uses a Random Forest classifier. We first discuss the datasets and the procedure that we use to extract the features, with examples from different devices. The evaluation is performed using both 10-fold cross validation and a split between training, validation and test-set. The latter is used for hyperparameter tuning. The results show that for reasonably large datasets the classifier achieves very high accuracy, as well as Precision and Recall rates. We further improve the performance on individual devices by selectively replicating the flows in the dataset, to achieve a better balance. We then evaluate a real-time implementation, and propose a runtime procedure to evaluate the model confidence level and trigger a retraining phase to adapt to a changing environment. A detailed analysis of the performance shows that the algorithm can support networks up to 100 Gbps on standard computing platforms.

INDEX TERMS Ensemble learning, FFT, IoT, random forest, spectral analysis, traffic classification.

I. INTRODUCTION

Progress in wireless network connectivity, miniaturization, and computing resources with advanced learning capabilities have seen the proliferation in the last decade of ubiquitous, independent and mostly autonomous devices dedicated to new and diverse applications, including sensing, remote control, fleet tracking, health care, and environmental monitoring and conditioning [1], [2]. This category of devices constitutes what is known as the *Internet of Things* (IoT) [3]. Network operators and component manufacturers must support this array of applications by providing network management functions such as resource planning, quality of service (QoS) provisioning, load balancing and lawful intrusion detection. At the same time, the diversity of IoT devices and applications results in infrastructure requirements that are vastly heterogeneous, posing significant challenges in delivering an optimized set of differentiated services. A key function and enabler for proper customization and network performance analysis is therefore the ability to automatically recognize and classify the nature of potentially a large number of communication flows in the network [4]. In this paper, we are interested in particular in distinguishing between traditional user traffic, such as e-mail, web surfing and media streaming, from traffic originating from IoT or other machine to machine communications.

IoT traffic could be very diverse in terms of network parameters and bandwidth requirements, reaching the very extremes of the available range. On one side, some devices may be monitoring quantities that change infrequently, such as a power meter or a temperature sensor. These devices would therefore connect to the network from time to time, and exchange minimal amount of data. The access pattern may be periodic, such as the power meter, or sporadic, triggered by the data itself, when the system is interested in detecting certain events. Latency in these cases is often of little concern, as long as it is reasonable. However, more and more often, a guaranteed connection may be required in order to quickly

The associate editor coordinating the review of this manuscript and approving it for publication was Najah Abuali.

trigger an alarm, such as the detection of hazardous events. On the other side of the spectrum, data may be streamed in large amounts, for instance by surveillance cameras or by telemetry systems on vehicles. Handling of the data is key, especially since things often carry quality of service requirements which are not easily supported by today's best effort communication networks [5]. One additional aspect is the rapidly increasing and large number of devices which may require the connection. Considering that communication between things may be less tolerant to latency, errors and failures than a corresponding human communication, and together with the needed service guarantees, which are not currently sufficiently supported by the network providers, service providers could be prompted to shift their communication products which need to evolve from a simple, undifferentiated dumb pipe to a more dedicated, performance aware network [6]. Traffic classification is therefore an essential feature to retrofit existing networks with devices that can support extra and smart functionalities. This could help at the infrastructure level, by supporting alternative routing decisions, or at the computational level, for job balancing at the server side. In addition, the value created by things is not limited to their specific functionality, but is more concentrated on the information that they produce. This information often acquires meaning when aggregated with the data produced by a multitude of devices. For these reasons, information processing does not generally occur on the individual devices, but is rather delegated to geographically distributed servers, known as *cloud services*, whether or not the device is technically able to handle the data. Communication is therefore not just limited to data which gets collected in a central repository, but is instead a key part of the functionality of a large integrated and distributed system. A network that is aware of the services can therefore take decisions that can help optimize the overall performance.

Detecting and classifying traffic can be accomplished using several techniques. *Deep packet inspection (DPI)* consists in observing the contents of the individual packets in order to detect characteristic patterns that identify the kind of protocol in use [7], [8]. The simplest methods work at the network level (TCP or UDP) and consider IP addresses and port numbers, which in most cases are individually assigned to particular services. More complete methods analyze also the higher levels of the protocol, up to the application layer, to avoid problems with port spoofing and to provide a more general approach. From the protocol, one can then deduce the class of service that is being used, and therefore classify the underlying application. An alternative approach is *deep flow inspection (DFI)*, which considers the overall behavior of the flow of packets, by analyzing, for instance, the size of the packets, their inter-arrival times, and other statistics. Deep flow inspection typically uses statistical classification and machine learning to provide a result. Because of the working principle of DFI, this method is also known as *behavioral classification*. Statistical and behavioral classifiers work at slightly different levels of abstraction, the first looking at features of the packets, the latter focusing more on the flow as a whole, looking at the access patterns. The benefits of statistical and behavioral classification are twofold. First, they can be applied whenever packets cannot be inspected, either because of the use of encryption or for privacy restrictions. In the second place, they may provide results also when applications use standard application level protocols, such as HTTP, to exchange information. Valenti *et al.* [9] provide a comprehensive review of statistical and behavioral methods, discuss the operation of support vector machines and decision trees, and analyze in detail both Kiss, a statistical classifier [10], and the Abacus behavioral classification algorithm [11].

In this paper we exploit a statistical flow classification, distinguishing between traditional and IoT related communication, and further discriminating among the IoT devices. To construct a classification procedure we use a supervised machine learning algorithm that can estimate the parameters for recognition from available labeled data. In previous work, we considered both clustering and decision trees [12] using features related to the fraction of upload versus download data. Clustering, however, did not produce sufficiently distinct classes when applied to real traffic, leading to confusion, while the decision tree has difficulty with generalization and suffers from overfitting. In this paper, we therefore use a different set of features, and focus instead on the length of the packets of a flow while adopting a more robust classification algorithm. In particular, we analyze the *spectral density* of the packet sizes of sequences of packets exchanged in a flow, intended as the communication between two particular parties, identified by their IP address/port pair. By doing this, we are able to account for both the absolute values of the packet sizes, as well as the sequence relationships of the packet exchange [13]. Spectral analysis thus provides information on the communication patterns, making the approach closer to behavioral classification. Besides, collecting packet lengths is extremely simple, with essentially no overhead, and does not require looking at the payload. Because the approach does not inspect the data, the method is applicable also in the presence of encryption. We train and evaluate the algorithm with a large dataset obtained from repositories opened to the public domain by their authors. The chosen classification algorithm is Random Forest, an algorithm that belongs to the class of *ensemble learning*, and which has been shown to perform well in this kind of problems [14]–[16]. In particular, Random Forest uses a set of simpler decision trees (100 in our case) to curb the overfitting problem. In addition, inference in Random Forest can be implemented very efficiently, using only simple comparison operations, an important characteristic in the context of high speed networks. To evaluate the classification algorithm we use the Weka framework [17], which provides the most popular machine learning methods. Our results show that given a sufficiently large dataset, the classifier achieves an accuracy of 98.0% with high reliability underscored by the high value of both Precision and Recall, and of the Matthews correlation coefficient. Our contribution includes the tuning of the tree depth, a study of the effect of changing the weight of each individual device in the dataset by replication, and the evaluation of the confidence level for real-time adaptation. Finally, we extensively evaluate the real-time performance of the algorithm, implemented natively in C, isolating the contribution of the individual computation steps, and studying its scalability when executed on a multi-threaded platform.

The paper is structured as follows. Section II explores related techniques in the area of statistical classification algorithms. We then give an overview of our approach in Section III, discussing the dataset preparation, and the feature computation. Section IV is dedicated to presenting the results of the evaluation. Finally, Section V provides a summary and outlines potential future directions.

II. RELATED WORK

Several methods have been proposed in the literature for packet classification, and we refer the reader to the existing literature for a systematic survey [18]. Here, we discuss the methods that are most closely related to our work.

The first aspect that must be considered when dealing with behavioral classification is an analysis of the communication pattern of the traffic that we want to classify. Shafiq *et al.* [19] have conducted a set of measurements to compare the machine-to-machine (M2M) traffic to traditional cellular smartphone traffic. The dataset comprises flows exchanged over the cellular data network in the USA. The first finding shows that M2M devices have a much larger uplink volume that downlink volume, in relative terms, compared to smartphones. This suggests a considerably different use of the network, and shows that M2M devices act more as content producers than consumers. The analysis also shows that M2M traffic follows business hours, and is significantly reduced in the weekends, as opposed to smartphone traffic which is virtually unchanged. Spectral analysis indicates strong periodicity in M2M traffic, corresponding to time intervals such as 1 hour, 30 minutes or 15 minutes, suggesting the timer-driven nature of M2M devices. Further analysis also reveals that devices are synchronized and coordinated. This may create congestion in the infrastructure. This frequency components are essentially absent from smartphone traffic. Sessions inter-arrival times are, instead, on average much longer for M2M devices than for smartphone traffic. An extensive list of discriminators for the purpose of flow classification is also presented by Moore *et al.* [20]. The dataset consists of general traffic captured for 24 hours at the authors' research facility. The report lists a total of 249 classes of discriminators, or features, that could be used by a classifier.

In our previous work, we have explored the use of several of these discriminators, including round-trip time and fractions of uplink vs. downlink volume, through a decision tree classifier [12]. In this paper, we adopt a different classification strategy that makes use of an ensemble of decision trees (a random forest) to reduce the overfitting problem. At the same time, we considerably reduce the number of features, and focus on only the spectral analysis of the packet length of the data flow.

Among the early applications, a few techniques were developed to discriminate between M2M traffic from remaining traffic (e.g., smartphones) specifically for cellular networks, using machine learning algorithms [21], [22]. These methods evaluate several features related to the traffic flow, including the number of packets, the data rate, the packet size, the inter packet-arrival time, as well as the IP address and TCP/UDP port numbers. Several derived features can also be computed from the flow data (averages, clusters, autocorrelations, etc.). Among the methods that are considered, there are unsupervised learning algorithms, which directly create classification, such as k-means, Expectation Maximization (EM) and Density-Based Spatial Clustering of Applications with Noise; and supervised methods, which require a learning phase with ground truth, such as J48, Naive Bayesian (NB) and Support Vector Machine (SVM). In our previous work, we have exploited neural networks as a supervised learning method [23]. Two interesting results can be learned from these studies. First, a limited number of features is sufficient to achieve a high level of classification accuracy. Second, decision trees (such as J48) appear to be the machine learning algorithms that perform best in the studies.

Sivanathan *et al.* present a rich deployment consisting of a smart environment instrumented with 28 IoT devices, such as cameras, sensors, lights and smart plugs [24], [25]. Traffic data was collected for six months, and then characterized in terms of several features, including protocols, data volume, port numbers and text patterns, activity and sleep cycles, and server queries. The authors present a two-stage hierarchical classifier, which uses a Naive Bayes Multinomial classifier to analyze domain names, port numbers and cypher suites, feeding a Random Forest classifier which integrates the remaining flow features. The combined approach is trained to recognize the different devices and shows a remarkable accuracy of 99.88%. A large part of our dataset is taken from this deployment, whose traffic data is made available in the public domain by the authors for download. Unlike this work, our aim is to classify traffic as traditional vs. IoT communication. More importantly, we focus on much fewer characteristics that capture the essence of an IoT device, rather than its specific implementation, and show that traffic patterns can be used as a distinguishing feature when analyzed in the frequency domain.

A similar approach is proposed by Meidan *et al.*, who measure a deployment of nine different IoT devices connected in a network together with two PC's and two smartphones [26]. The authors analyze features from the network, transport and application protocol layers, and classify the devices based first on a single communication session, and then on multiple sessions, outlining how the thresholds of the binary classifiers can be optimized to improve performance. While the reported classification accuracy is high (in excess of 99%), the authors do not discuss the way the features were selected, nor the kind of binary classifier and the way it is trained. This makes a comparison with other approaches problematic. As in the previous case, in our work we choose to ignore the protocol parameters to focus on the behavioral characteristics of the communication pattern, to abstract from the particular device.

An interesting approach is presented by Lopez-Martin *et al.*, who apply a combination of a recurrent neural network (RNN) with a convolutional neural network (CNN) to perform traffic classification on a large dataset extracted from the Spanish academic backbone network [27]. For every flow, the authors consider the first 20 packets and collect a number of high-level header-based features, such as ports, window size, payload size and inter-arrival time. The main difference with respect to the previous approaches is that the features are organized in a time series, which is applied to the convolutional neural network as if it were an image, to identify local correlations. They then analyze the importance of each feature, by looking at the classification performance as they are added to or removed from the set. Overall, an accuracy up to 96% is reported for this work, on a dataset with over 100 different classification labels. In a different study, Yang *et al.* exploit a Conditional Variational Autoencoder to address the problem of imbalance in intrusion detection systems, and use a 6-layer Deep Neural Network for classification [28]. As explained previously, our approach differs in the kind of features that we select from the dataset, as we ignore all the header data. Finally, Iliyasu and Deng develop a semi-supervised method using Deep Convolutional Generative Adversarial Networks [29]. In general, they achieve lower accuracies on the classification problem relative to other works, however they can work with small labeled datasets. In addition, they require flows with long duration, which is not usually the case in the IoT. In general, a trend is developing towards using deep networks to learn features, especially in the area of intrusion detection [30]. Wang *et al.* provide a survey of deep learning for traffic classification, and also discuss the problem of data imbalance [31].

A number of solutions make use of ensemble learning to avoid overfitting and enhance the generalization power of the model. For instance, Shahid *et al.* extract features such as packet size and inter-arrival times of the first packets (10) of a flow from the network traffic of a smart home equipped with four devices [15]. The authors evaluate six different classification algorithms, and show that Random Forest performs best with an accuracy as high as 99.9%. One limitation of this study is the low number of devices and the use of data from the same deployment for both training and test, in the absence of non-IoT traffic. More recently, Amouri *et al.* have also used a Random Forest classification algorithm in the context of intrusion detection systems [32]. Thangavelu *et al.* perform a similar study, focusing on scalability and the ability of the classifier to dynamically identify new devices [14]. This is accomplished by clustering the flows first with a standard k-means algorithm, then using a distributed semi-supervised clustering method by aggregating features. Clustering uses features such as DNS queries, number of packets, activity period in a session, TLS packet length, flow duration, and number of packets of distinguished protocols such as DNS. The clusters are used as aggregate features to then periodically train a supervised learning algorithm for the final classification, which can therefore learn models for new devices. The approach is evaluated on an experimental setup consisting of 16 IoT devices for smart homes, monitored over a period of one week. The results show that a random forest classifier performs better than k-NN and Gaussian and Bernoulli naive Bayes with a 98% accuracy.

Pinheiro *et al.* present an approach based on a reduced set of features, relying essentially on the length of the packets (and their statistics, such as mean, mode and standard deviation) seen in a 1-second sampling window of the flow, to perform classification and reduce latency [16]. The selected features are independent of the specific header fields, so that classification can operate also in the presence of encryption. Classification is performed in three stages, distinguishing IoT and non-IoT traffic first, followed by the IoT device identification, and by the specific device event. Among five different classifiers, including k-NN, Random Forest, Decision Tree, SVM and Majority Voting, Random Forest is shown to perform best with accuracy reaching 96%. The approach is evaluated on a testbed composed of three IoT devices, complemented by traffic from the mentioned dataset of Sivanathan *et al.* [25]. Our approach is similar in spirit to this work, in that we rely on fewer attributes that characterize the behavior of the communication pattern, rather than the protocol data. Our novel contribution lies in the use of the frequency domain as an alternative space to perform the classification.

Finally, in the context of traffic offloading, Han *et al.* propose a method of traffic classification based on the Fast Fourier Transform (FFT) of the values of the first six bytes of the application payload [33]. This has the advantage that a single packet is sufficient to proceed with classification. On the other hand, encryption, compression and data obfuscation renders this approach not applicable, as the content of the packets becomes randomized, and therefore no longer characteristic of one class of communication.

III. SYSTEM OVERVIEW

In our approach, we construct a set of identifying features by computing the FFT spectrum of the series of packet lengths of a communication flow in the frequency domain. The basis of this technique was introduced by Liu *et al.*, who consider the problem of classifying application traffic in the context of an encrypted network [13]. In our case, we apply the technique to distinguishing between IoT and non-IoT traffic, rather than to generic applications, and refine the method by considering alternatively the client to server, the server to client or the inter-arrival time as the source of data. This overcomes problems when the application does not transfer data, but only produces keep-alive messages.

Among other approaches based on the frequency domain, Tegeler *et al.* use the FFT of a sampling of the traffic in time, interpreted as a binary signal (1 when active, 0 when not active), as one of different features used to detect malware in a network [34]. In the context of IoT networks, given that packet timings may vary by several orders of magnitude, finding the right trade-off for the sampling period is particularly problematic, making this method difficult to apply. In addition, depending on where the data is captured, packet timings may well be affected by the congestion of the network, and therefore by the activity of other unrelated devices, and is subject to potentially intense variability. Zhou and Lang adopt a similar approach, extending the time series to considering packet rate, number of distinct IP addresses and the inter-arrival times [35], with the aim of detecting network intrusions by looking at regularities. From this study, which was applied only to synthetic intrusion detection datasets, we borrow the idea of considering features other than the payload size.

The following subsections describe our approach in detail.

A. BACKGROUND

Before presenting the system architecture, we briefly review the technologies that we have used in our implementation.

Flows are collected into `.pcap` files, for *packet capture*, which contain a sequence of packets annotated with a timestamp with nanosecond precision that defines their transmission time, and their length in byte. Packets are stored in binary format, as a sequence of bytes, including all the protocol headers. A `.pcap` can be generated by any of several network monitoring software, such as tcpdump or Wireshark, which listen to the network and store all packets that are transferred. Software can be written to read and write this format using the standard API provided by several library implementations, such as libpcap or WinPcap. Our scan and check phase, described below, were developed using this infrastructure.

The data extracted from the `.pcap` files are processed by a Fast Fourier Transform to generate the features used by the classification algorithm. There exist several implementations of the FFT algorithm, which differ for flexibility, optimization and performance. We have used the KISS FFT library[1] during the training and test phase, because of the simplicity of integration into the data preparation flow, and the FFTW v3.3.8 library[2] because of its high performance for the C implementation used during real-time analysis.

There is a number of frameworks that can be used to build and train classifiers. In this work we have used Weka [17] for experimenting and testing. Weka is a graphical tool that combines classification algorithms and visualization methods, making it suitable for model development. In addition, it provides extensive statistical reports on the trained model, implements k-fold cross validation, and computes accuracy metrics that make it easy to estimate the classification performance of the model, and tune its hyper-parameters. For the actual implementation we have instead used the SciKit-Learn framework [36], which gives access to its internal data structures and simplifies the process of generating optimized code suitable for deployment.

The Random Forest classifier that we use is built as a collection of decision trees. Each tree node inspects the value of a feature of the input, and chooses one branch depending on a threshold determined during training. Branches separate the input space, clustering together samples with similar features. A classification is performed when reaching a leaf of the tree. In a random forest, several decision trees concur equally to the final classification, obtained by majority voting, in what is known as *ensemble learning*. Because the individual trees of a random forest are simpler than a single monolithic large decision tree, the random forest can more easily generalize and suffers less from the overfitting problem, in which the classifiers overly adjusts to the training examples.

B. DATASET PREPARATION

Our analysis is based on a number of network captures codified as streams of packets encoded in standard `.pcap` files which are subsequently filtered by our software and analyzed by machine learning tools. Our main source of data is provided by a large dataset made available[3] by the aforementioned work of Sivanathan *et al.* [24], [25]. The data is obtained by capturing the traffic of a deployment of which we include 21 IoT devices and 7 non-IoT devices, shown in Table 1. We refer to this dataset as the *Australia dataset*. The capture spans several days of operation.

TABLE 1. **Australia dataset: Device list from data by Sivanathan *et al.* [24], [25].**

IoT Device	IoT Device
Smart Things	Withings Smart Baby Monitor
Amazon Echo	Belkin Wemo switch
Netatmo Welcome	TP-Link Smart plug
TP-Link Day Night Cloud camera	iHome
Samsung SmartCam	Belkin wemo motion sensor
Dropcam	NEST Protect smoke alarm
Insteon Camera	Netatmo weather station

IoT Device	non-IoT Device
Withings Smart scale	Android Phone
Withings Aura smart sleep sensor	Laptop
Light Bulbs LiFX Smart Bulb	MacBook
Triby Speaker	Android Phone
PIX-STAR Photo-frame	IPhone
HP Printer	MacBook/Iphone
Nest Dropcam	Samsung Galaxy Tab

Additional data is obtained from a study conducted by Guo and Heidemann, who classify IoT flows based on the IP addresses contacted by each devices, in the context of the USC/LANDER project [37]. The procedure requires a careful

[1]KISS FFT, available at https://github.com/mborgerding/kissfft
[2]http://fftw.org/

[3]The dataset can be downloaded at the following address: https://iotanalytics.unsw.edu.au/iottraces

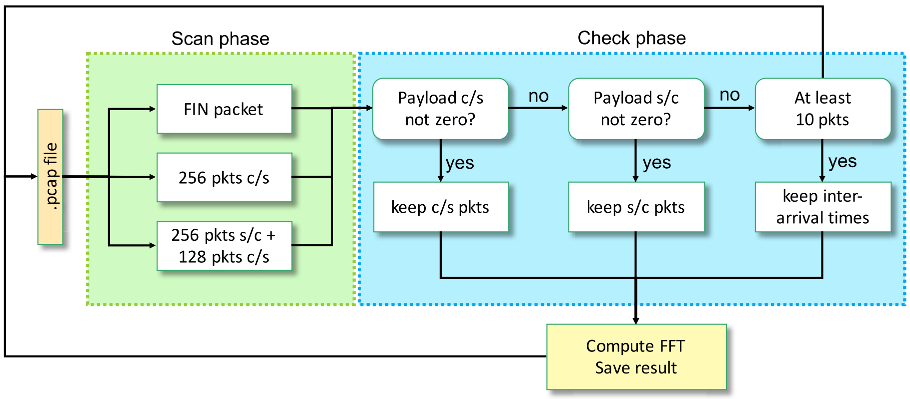

FIGURE 1. Data acquisition phases and features computation.

filtering of the addresses and setting appropriate thresholds on the number of accesses to avoid false classifications. From this dataset, we extract traffic for 16 IoT devices, shown in Table 2, which were captured from a College Campus network.[4] Although some of the devices are similar to those found in the first dataset, we do label them differently to account for potential different behaviors related to their specific use. In the following, we refer to this dataset as the *California dataset*.

TABLE 2. California dataset: Device list from data by Guo and Heidemann [37].

IoT Device	IoT Device
HP Printer	TP-Link Smart plug
Amazon Dash Bounty Button	Foscam IP CAM2
Amazon Echo	Google Smart Speaker
Amazon Fire SmartTVStick	Philips-Hue
AMCREST IP CAM	RENPHO Humidifier
Belkin Wemo switch	TENVIS IP cam
D-link IP CAM	TP-Link SmartLightBulb
Foscam IP CAM	Wize IP CAM

To add variety to the dataset, we also include simulated flows generated by an IoT device software simulator called Mimic IoT Simulator [38]. The academic license of the software (version 17.10) we have used allows one to simulate up to 250 concurrent sensors, divided in two main brands and configurations from Intel and Bosch. In our analysis, the two types of sensors are considered as a single class. Using different configuration files, one can try to make the network as diverse as possible, simulating different sensors and periodicities. Nevertheless, the behaviors will be somewhat homogeneous, as it is difficult to model event-triggered sensors in this framework. The data is captured by instructing the simulator to contact an MQTT broker and setting up an MQTT client, such as `mosquitto`, that subscribes to the data and receives the live messages from the broker, closing the loop.

[4]Access to the dataset can be requested at the following address: https://ant.isi.edu/datasets/requests.html

Overall, the total number of flows collected from each device is shown in Table 3, limited to the IoT devices. As far as non-IoT flows is concerned, we are using 19,187 flows from the Australia dataset. To these, we add 2,313 flows from an experiment from the University of New Brunswick [39],[5] and 10,335 flows from a dataset collected by the Network Monitoring and Measurements research group at the University of Napoli [40], [41].[6] Finally, we also include the "bigFlows" traffic dataset from Appneta Tcpreplay with 13,847 flows [42].[7] In total, there are 45,682 non-IoT flows.

C. FEATURES COMPUTATION

The procedure we follow to create the set of features is shown schematically in Fig. 1. The data is processed directly from the `.pcap` file using a dedicated software built on top of `libpcap`.[8] The analysis starts from a scan phase in which we identify and break up the flows, a check phase in which we select the packets, and a computation phase in which we compute the Fourier Transform of the data series. More specifically, in the first phase, we *scan* the flows to determine when to start the computation. The scan operation works through the `.pcap` file considering one packet at a time. The header is used to extract the timestamp and the length of the packet. The actual protocol headers in the packet data are used to extract flow information, in order to group packets into *flows* according to their source and destination, or client and server. A flow is constructed from the IP addresses and port number, and the TCP flags, where the SYN flag denotes the beginning of the flow, and the FIN flag denotes its end. Each flow is forwarded to the next phase of computation whenever we reach a FIN packet (which identifies the end of the flows), or whenever we scan 256 packets from client to server, or if we scan 256 from server to client while at the same time we see at least 128 packets from client to server.

[5]Available at https://www.unb.ca/cic/datasets/vpn.html
[6]Available at http://traffic.comics.unina.it/Traces/ttraces.php
[7]Available at https://tcpreplay.appneta.com/wiki/captures.html
[8]The code is available at https://github.com/gencir94/Libpcap-spectral-analysis

TABLE 3. Number of collected flows for each device.

IoT Device	Flows	IoT Device	Flows
Smart Things	32	Amazon Echo	3095
Netatmo Welcome	1809	TP-Link Day Night Cloud camera	851
Samsung SmartCam	5368	Dropcam	256
Insteon Camera	2998	Withings Smart Baby Monitor	4333
Belkin Wemo switch	6288	TP-Link Smart plug	167
iHome	149	Belkin wemo motion sensor	62440
NEST Protect smoke alarm	67	Netatmo weather station	1659
Withings Smart scale	28	Withings Aura smart sleep sensor	2532
Light Bulbs LiFX Smart Bulb	29	Triby Speaker	137
PIX-STAR Photo-frame	818	HP Printer	63
Nest Dropcam	359	HP Printer California	4
TP-Link Smart plug California	39	Amazon Dash Bounty Button	8
Foscam IP CAM2	87	Amazon Echo California	639
Google Smart Speaker	185	Amazon Fire SmartTVStick	396
Philips-Hue	261	AMCREST IP CAM	337
RENPHO Humidifier	5	Belkin Wemo switch California	45
TENVIS IP cam	1	D-link IP CAM	331
TP-Link SmartLightBulb California	4	Foscam IP CAM	1035
Wize IP CAM	44	MIMIC	6946
Total	**103845**		

The reason we do this is that we have found experimentally that the packets going from the client to the server provide a higher distinguishing power than the reverse direction. This is because, as noted earlier, for IoT devices the uplink traffic is much higher than the downlink traffic [12], [19], and further carries more information. In practice, in many cases the server simply acknowledges the reception of a packet, without any additional data, making the recognition hard as this behavior is invariant across different devices. Note also that flows longer than 256 packets are broken up into different samples of length 256. This has the advantage of reducing the latency to obtain the classification results as well as its computational complexity. Conversely, flows that do not reach 256 packets at the FIN are padded with zeros to reach the desired length, and processed immediately. This is necessary, as the FFT algorithm requires always the same number of input values.

The sample size, i.e., the maximum number of packets in a flow to be considered for classification, 256 in our case, is a parameter of the classification algorithm. This number is a compromise between the rate at which we have to run the classification algorithm, the spectral resolution, and the amount of information collected in a sample, and depends also on the statistics of the flows. What we have found is that flows *of IoT devices* in our dataset typically have in the order of 20 packets per flow, while the average size of the packets is around 365 bytes, including all headers and the interpacket gap. This seems to indicate that much fewer packets than 256 would be sufficient in the analysis. On the other hand, reducing the number of packets, for instance to only 32, has considerable disadvantages. First, it does not reduce the latency of the classification of short flows, as flows are classified immediately upon receiving the FIN flag, by padding the sequence with 0's. Second, it does reduce the spectral resolution, in our example by a factor of 8. But more importantly, it increases the number of classifications to be performed by almost a factor of 8. This is because, while IoT flows are typically short, the statistics for non-IoT devices are quite different, and flows with *hundreds* or even *thousands* of packets can be quite common and prevalent in the network. We refer in particular to a study by Jurkiewicz *et al.* who analyze the flow length and size distributions of internet traffic in a campus environment [43]. Their findings show that while the majority of flows have in fact very few packets (i.e., 95% of the flows have fewer than 100 packets), the flows with lots and large packets dominate the traffic: according to their data (see Figure 2 in ref. [43]) 85% of the traffic is carried by flows with more than 1,000 packets, and more than 70% by flows with more than 10,000 packets. The same (in fact even more) is true in terms of the amount of data. Our choice thus reduces the number of classifications that must be performed per second, without sacrificing latency. Nevertheless, this parameter may be adjusted to the specific requirements and traffic patterns of the network.

During the second phase, we *check* the length of the packets for information content. For length, we denote the size of the payload, excluding the base protocol headers (IP, TCP and UDP). If the sequence of packets from client to server consists of only empty payloads, we check the packets in the reverse direction. If these are also empty, then we use the packet inter-arrival times, computed from their timestamps, of the client to server packets as the data, since the payload provides no useful information, while the periodicity is used as a discriminator.

The FFT is computed on the selected data by a dedicated software library and produces a symmetric spectrum

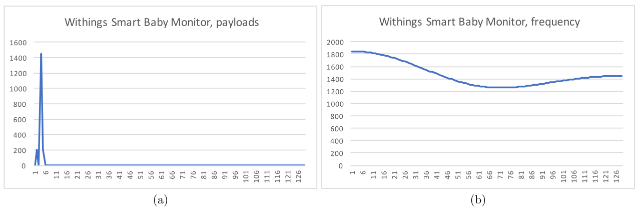

FIGURE 2. **Withings Baby monitor. (a) Payload time series. (b) Frequency series.**

of 256 elements, of which only the right 128 values are retained. The computation follows the traditional formulation

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N},$$

where x_n are the selected packet payload lengths in the flow, N is the number of samples (256 in our case), and X_k are the spectrum output coefficients. In particular, we use the magnitude $||X_k||^2$ of the spectrum, which is proportional to the power content at each specific frequency. The computation of the spectrum resembles the application of the first layers of a convolutional neural network, except that the filter are fixed and not learned from the data. This, on the other hand, simplifies both training and inference.

Having characterized each flow with its FFT, we determine the ground truth label of the data by matching the Ethernet MAC address of the packets to those of the devices present in the network. Notice how we cannot use the MAC address in the classifier to identify flows, as the MAC address characterizes the sender of the *physical* segment of the network, which could be the router on behalf of the device. Instead, we must use the IP addresses in the classifier as explained earlier. In fact, we have end-to-end visibility in our traces for determining the ground truth, something that the classifier does not generally have. The FFT coefficients and the labels are then assembled into an `.arff` file that is given as input to Weka [17], which in turn partitions the data into training, validation and test sets and performs the model optimization. At the end of the process, the tool provides the performance metrics from cross validation and testing, as discussed in the next section.

IV. RESULTS

In this section, we summarize the results obtained by training a Random Forest classifier with the data acquired using the procedure outlined in the previous section. For training and evaluation we use the Weka framework [17] which provides several classification algorithms and an environment suitable for validation. In particular, we evaluate the classification accuracy using 10-fold cross validation. With this method, the dataset is divided into 10 random subsets, which are alternatively used for training and for determining the classification accuracy. The results from ten different rounds are then averaged to give the final metrics. Because our dataset is somewhat skewed towards the IoT class, an evaluation based on accuracy alone, i.e., the ratio between the correctly classified flows and the total number of flows, is unable to provide a proper picture of the performance of the classifier. For this reason, in addition to True Positives, False Positives and False Negatives, we make use of the following metrics [44]:

- Precision: for a given class, it is the ratio of its True Positives and the sum of True Positives and False Positive (i.e., a sample of another class that is labeled as one of this class).
- Recall: for a given class, it is the ratio of the True Positives and the sum of True Positives and False Negatives (i.e., a sample of this class is labeled as not of this class).
- Matthews correlation coefficient (MCC): the MCC returns a value between -1 and $+1$, where $+1$ represents a perfect prediction, 0 is no better than a random prediction and -1 indicates total disagreement between prediction and observation.

The Matthews correlation coefficient measure is particularly significant for binary classification (for instance, when we distinguish between IoT and non-IoT classes) especially when the classes are of different size as in our case. In addition to these, one alternative measure is the F-Measure, a widely used metric in classification, which weighs both Precision and Recall in a single metric by taking the harmonic mean: $2 \times$ Recall $\times$ Precision / (Recall + Precision). When running the 10-fold validation procedure, the Weka tool automatically returns these measures averaged over the different iterations of the process. In the tables in Section IV-B, for lack of space we do not show the F-Measure explicitly as this can be easily deduced from Precision and Recall.

A. DATA VISUALIZATION

It is useful to visualize the data features that correspond to the packet flows of different devices, to appreciate the distinguishing power of the transformed signal. In this section we discuss a few examples, where we show and compare the time series as well as the frequency spectra of selected flows.

Fig. 2 to Fig. 8 show both the series of the payload (on the left) and its FFT, limited to one side of the symmetric magnitude of the spectrum (on the right). We see that for short flows, the ''energy'' of the packets is spread across the entire spectrum with a shape that depends on the number of peaks that are found in the time series. These cases are shown in Fig. 2 and Fig. 3. The energy in the spectrum, i.e., the magnitude of each frequency component, depends on the size of the payloads in the flow, and this constitutes another difference than can be used by the classifier to distinguish the flows. This is true in all the examples shown.

Fig. 4 and Fig. 5 show the spectrum of somewhat longer payload series, with limited regularities. These kind of spectra are also spread out, but are more concentrated around certain specific frequencies that distinguish the nature of the traffic. On the other hand, traffic that is characterized by strong regularities gives rise to precise peaks in the corresponding spectrum. This case is shown for instance in Fig. 6, where the

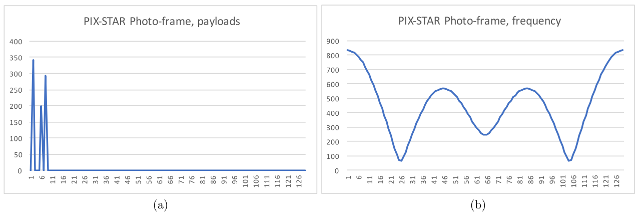

FIGURE 3. PIX-STAR Photo Frame. (a) Payload time series. (b) Frequency series.

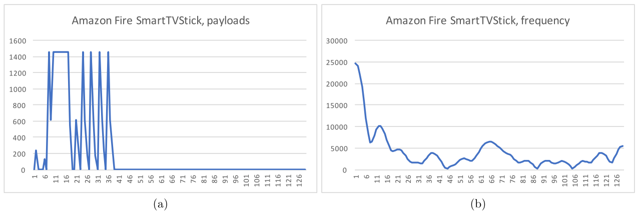

FIGURE 4. Amazon Fire SmartTVStick. (a) Payload time series. (b) Frequency series.

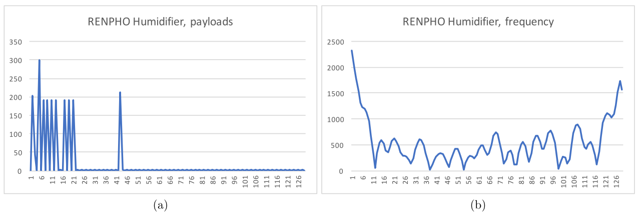

FIGURE 5. RENPHO Humidifier. (a) Payload time series. (b) Frequency series.

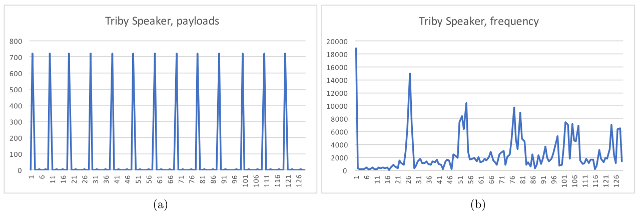

FIGURE 6. Triby Speaker. (a) Payload time series. (b) Frequency series.

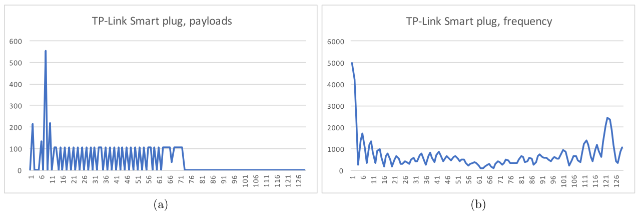

FIGURE 7. TP-Link Smart Plug. (a) Payload time series. (b) Frequency series.

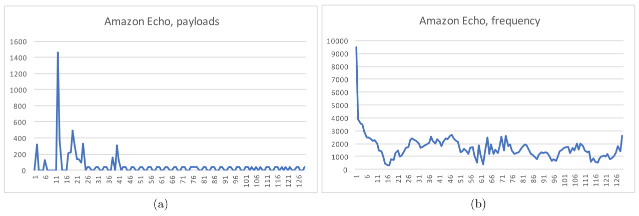

FIGURE 8. Amazon Echo. (a) Payload time series. (b) Frequency series.

spectrum shows peaks at very specific frequencies, on top of a noisy background that depends on the small variabilities in the data. The shape of the spectrum is clearly very different compared to the previous cases.

Finally, longer payload series with recurring and closer regularities result in denser peaks in the spectrum, as shown in Fig. 7 and to a lesser extent in Fig. 8. This last example is more difficult to characterize in terms of a definitive shape, but at the same time can be easily set apart from the more distinctive patterns exhibited by the other devices.

B. CLASSIFICATION RESULTS

In this section we present the results of the classification algorithm evaluated using the 10-fold cross validation technique. For classification, we have used a Random Forest classifier. This classifier is an example of an *ensemble* algorithm: instead of a giant decision tree, which may easily overfit the data, the Random Forest is composed of a number of simpler decision trees, each providing its own classification. The overall results corresponds to the majority voting of the individual trees. Because the trees are simpler, they also result in a simpler hypothesis function, reducing the overfitting problem. In our case, we have instructed the Weka tool to construct a Random Forest with 100 trees, each with unlimited depth. The number of trees is another parameter of the classification algorithm. Our choice is an attempt to balance the algorithm runtime complexity with its ability to generalize and give information on the quality of the classification. Specifically, we compute the confidence in the classification as the fraction of trees that compose the majority. The higher the number of trees, the better the resolution. In Section IV-F we use a threshold of 0.9 to trigger a retraining, therefore we opted for a resolution of at least 0.01. There are studies in the literature that analyze how accuracy changes with the number of trees. Oshiro *et al.* conclude that between 64 to 128 trees provides the best trade-off using datasets from the biomedical domain [45]. Experiments with our dataset show that 50 trees decrease accuracy by only a few percentage points. We do not however present an analysis of this parameter, which could easily be adjusted depending on the specific situation and application. On the other hand, Section IV-D explores the impact of bounding the depth of the trees.

We have divided the evaluation into two main parts, each composed of three different mixes of the dataset. In the first part, we evaluate in particular the ability of the classifier to distinguish among the different IoT devices. Each flow in the dataset is therefore labeled with the device name, and the classifier is trained to distinguish the individual labels. In a second set of experimental evaluation we introduce also the traditional non-IoT traffic. In this case, we evaluate both a classifier that is able to distinguish among the different devices, as well as a classifier that simply distinguishes

TABLE 4. Classification performance for the Australia dataset, IoT only.

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
Dropcam	235	21	91.8%	0.0%	88.0%	91.8%	89.9%
Smart Things	23	9	71.9%	0.0%	100%	71.9%	84.8%
Withings Smart baby monitor	4328	5	99.9%	0.0%	99.9%	99.9%	99.9%
Belkin Wemo Motion Sensor	62413	27	100.0%	1.6%	99.2%	100.0%	98.7%
Samsung SmartCam	5357	11	99.8%	0.0%	99.4%	99.8%	99.6%
Belkin_Wemo_Switch	6253	35	99.4%	0.0%	99.7%	99.4%	99.5%
PIX-STAR Photo frame	812	6	99.3%	0.0%	99.3%	99.3%	99.3%
Amazon_Echo	3028	67	97.8%	0.1%	98.0%	97.8%	97.8%
TP-Link Smart Plug	166	1	99.4%	0.0%	96.5%	99.4%	97.9%
Netatmo weather station	1657	2	99.9%	0.0%	99.9%	99.9%	99.9%
TP-Link DayNight CloudCam	819	32	96.2%	0.0%	99.4%	96.2%	97.8%
Netatmo Welcome	1796	13	99.3%	0.0%	98.5%	99.3%	98.9%
Withings Smart Scale	26	2	92.9%	0.0%	100%	92.9%	96.4%
Triby Speaker	122	15	89.1%	0.0%	93.8%	89.1%	91.4%
NEST Protect smoke alarm	60	7	89.6%	0.0%	100%	89.6%	94.6%
HP printer	61	2	96.8%	0.0%	98.4%	96.8%	97.6%
Insteon Camera	2998	0	100%	0.0%	99.8%	100%	99.9%
Withings AuraSmartSleepSensor	2062	470	81.4%	0.0%	99.1%	81.4%	89.6%
iHome	139	10	93.3%	0.0%	98.6%	93.3%	95.9%
Light Bulbs LiFX SmartBulb	23	6	79.3%	0.0%	95.8%	79.3%	87.2%
Nest Dropcam	336	23	93.6%	0.0%	93.1%	93.6%	93.3%
Weighted Avg.	**92714**	**764**	**99.2%**	**1.1%**	**99.2%**	**99.2%**	**98.6%**

between the IoT and the non-IoT class. The following two sections present the details of the evaluation.

1) CLASSIFICATION OF INDIVIDUAL DEVICES

We train three different classifiers to evaluate the classification performance on different datasets. In the first two cases we look at the classifier using the Australia and the California datasets individually. The third classifier is instead trained on the combined dataset, including the simulated flows from the Mimic simulator. The results for these experiments in terms of Precision, Recall and MCC are shown in Tables 4, 6 and 8, while Tables 5, 7 and 9 show the corresponding accuracy.

The results are especially positive for the Australia dataset, which comprises the majority of the flows. In particular, the False Positive rate is very low, and the MCC is close to 1, indicating a high degree of reliability (see Table 4). The overall accuracy for this dataset come in excess of 99% (Table 5).

The California dataset, which is much smaller in size, gives a mixed outcome when considered alone. The break-up of the classification results in Table 6 shows that a few of the devices are difficult to recognize, especially when the corresponding number of flows in the training set is particularly low. In this case, the 10-fold cross validation may at times fail to include the flows in the actual training set. This indicates that the classifier needs a sufficient number of examples to correctly identify devices which have a low utilization of the network. The overall accuracy, in this case is much lower at 86.52% and the MCC only achieves a value of 84.2%. We will address this shortcoming in Section IV-E by replicating the flows.

When combined (Table 8 and 9) the overall classification performance is acceptable, although the data is obviously dominated by the much larger Australia dataset. In this case, we observe a slight increase in False Positive rate, although the weighted average fails to show this because the weight of each device, especially those for which recognition is harder, is much lower with the large size of the dataset. The tables also show that the algorithm performs extremely well on the Mimic simulated flows. These flows are clearly more homogeneous, indicating that traffic captured from real devices is essential in the context of classification.

2) TRADITIONAL VS. IoT TRAFFIC CLASSIFICATION

In this set of experiments we have trained the classifier to recognize the IoT devices together with the non-IoT traffic lumped into a single class. The addition of the non-IoT traffic slightly degrades the classification accuracy. Table 10 shows the results, which indicate that the non-IoT flows can be well separated from the individual devices. The overall accuracy is $137133/(137133 + 2027) = 98.54\%$.

The experiment is repeated by giving the IoT devices a single label, to discriminate between traditional and IoT traffic using binary classification. The results for the Australia dataset are shown in Table 11. The classification achieves an accuracy of 99.0%, proving the high performance that can be obtained by applying this method.

The same experiments are conducted on the California dataset, with the results shown in Table 12 and 13. Similarly to the previous experiments, the accuracy is lower in this case. Nevertheless, the classifier is still able to distinguish the non-IoT flows well, giving an overall accuracy of 96.15% when the classifier distinguishes also the individual devices. The improvement however is only apparent. In fact, clearly the accuracy has improved because of the non-IoT flows,

TABLE 5. **Australia dataset IoT only: overall accuracy.**

Correctly Classified Instances	InCorrectly Classified Instances	Total Number of Instances
92714 - 99.18%	764 - 0.82%	93478 - 100%

TABLE 6. **Classification performance for the California dataset, IoT only.**

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
HP printer	2	2	50.0%	0.1%	50.0%	50.0%	49.9%
Amazon_Dash_Bounty_Button	8	0	100.0%	0.0%	100.0%	100.0%	100.0%
Amazon_Echo	622	17	97.3%	7.50%	74.8%	97.3%	81.6%
Amazon_FireTVStick	200	196	50.5%	2.2%	74.6%	50.5%	57.5%
AMCREST_IPCAM	284	53	84.3%	1.3%	87.4%	84.3%	84.3%
Belkin_Wemo_Switch	30	15	66.7%	0.2%	78.9%	66.7%	72.2%
D-link_IPCAM	304	27	91.8%	0.8%	92.1%	91.8%	91.1%
FOSCAM_IPCAM	978	57	94.5%	1.5%	96.5%	94.5%	93.6%
FOSCAM_IPCAM_vers2	52	35	59.8%	0.3%	83.9%	59.8%	70.2%
Google_Smartspeaker	169	16	91.4%	1.1%	82.8%	91.4%	86.2%
Philips_Hue	249	12	95.4%	0.4%	95.4%	95.4%	95.0%
RENPHO_humidifier	0	5	0.0%	0.0%	-	0.0%	-
TENVIS_IPCAM	0	1	0.0%	0.0%	-	0.0%	-
TP-Link Smart Plug	32	7	82.1%	0.2%	84.2%	82.1%	82.9%
TP-Link SmartLightBulb	2	2	50.0%	0.1%	50.0%	50.0%	49.9%
Wyze_IPCAM	28	16	63.6%	0.1%	84.8%	63.6%	73.2%
Weighted Avg.	**2960**	**461**	**86.5%**	**2.4%**	**86.6%**	**86.5%**	**84.2%**

TABLE 7. **California dataset IoT only: overall accuracy.**

Correctly Classified Instances	InCorrectly Classified Instances	Total Number of Instances
2960 - 86.52%	461 - 13.48%	3421 - 100%

while the recognition of the devices has obviously worsen. In particular, several IoT flows are classified as non-IoT, producing a large increase of the False Positive rate of the non-IoT class.[9] This is highlighted by the value of the MCC, whose weighted average (excluding the devices for which the computation does not provide a result) at 68.9% is much lower than in the case of the Australia dataset.

The binary classification results for the California dataset are shown in Table 13. Again, while accuracy is high, the MCC is much lower at 70.6% due to the high False Positive rate.

Our final results considering the entire dataset, including the simulated flows, are shown in Table 14 and 15. The accuracy for the recognition of the individual flows reaches 97.37% with an MCC of 96.5%. This can be considered a good result, considering the large number of flows in the dataset.

The results of binary classification with the entire dataset is finally shown in Table 15, with an accuracy of 98.0% and an overall MCC of 95.3%, highlighting again the reliability of the classification algorithm.

[9]These values can be extracted from the confusion matrix produced by Weka. We do not show this data explicitly, as the matrix would be too large to display, and we instead quote the summary value.

C. TEST-SET EVALUATION

The above results were obtained through 10-fold cross validation. Because we leave the parameters of the models unchanged, this effectively partitions the dataset into a training and a test set, averaging the results across the different folds. In the following section, we will use the cross-validation technique to tune the depth of the trees. In this case, we must set aside part of the dataset as a proper test set, which is not used during the tuning operation, to avoid overestimating the classifier performance. We opt for a 70%-30% split between training (used also in cross-validation) and test set, chosen randomly before applying the entire procedure. For sanity check, we have run the training procedure and test evaluation separately, without 10-fold cross validation, to verify that we obtain results that are consistent with those reported in the previous section. Indeed this is the case. For brevity, we report only the results for the combined dataset including the non-IoT flows, for binary classification, shown in Table 16. The absolute values of the flows are obviously much smaller, as they correspond to only 30% of the dataset, as it can be readily verified. However, the performance is in line with that shown previously in Table 15, with differences in the order of one tenth of a percent. The results for the individual datasets, although not shown, are likewise similar to those of the 10-fold cross validation.

TABLE 8. Classification performance of the combined dataset, IoT only.

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
Dropcam	235	21	91.8%	0.0%	88.0%	91.8%	89.9%
Smart Things	27	5	84.4%	0.0%	100%	84.4%	91.9%
Withings Smart baby monitor	4327	6	99.9%	0.0%	99.6%	99.9%	99.7%
Belkin Wemo Motion Sensor	62413	27	100.0%	1.3%	99.2%	100.0%	98.9%
Samsung SmartCam	5355	13	99.8%	0.0%	99.3%	99.8%	99.5%
Belkin_Wemo_Switch	6252	36	99.4%	0.0%	99.7%	99.4%	99.5%
PIX-STAR Photo frame	810	8	99.0%	0.0%	98.1%	99.0%	98.5%
Amazon_Echo	3021	74	97.6%	0.2%	95.1%	97.6%	96.2%
TP-Link Smart Plug	166	1	99.4%	0.0%	96.5%	99.4%	97.9%
Netatmo weather station	1656	3	99.8%	0.0%	99.4%	99.8%	99.6%
TP-Link DayNight CloudCam	818	33	96.1%	0.0%	99.2%	96.1%	97.6%
Netatmo Welcome	1796	13	99.3%	0.0%	97.4%	99.3%	98.3%
Withings Smart Scale	23	5	82.1%	0.0%	100%	82.1%	90.6%
Triby Speaker	121	16	88.3%	0.0%	87.7%	88.3%	88.0%
NEST Protect smoke alarm	61	6	91.0%	0.0%	98.4%	91.0%	94.6%
HP printer	57	6	90.5%	0.0%	95.0%	90.5%	92.7%
Insteon Camera	2998	0	100%	0.0%	99.8%	100%	99.9%
Withings AuraSmartSleepSensor	2045	487	80.8%	0.0%	97.6%	80.8%	88.5%
iHome	136	13	91.3%	0.0%	97.8%	91.3%	94.5%
Light Bulbs LiFX SmartBulb	18	11	62.1%	0.0%	94.7%	62.1%	76.7%
Nest Dropcam	327	32	91.1%	0.0%	90.6%	91.1%	90.8%
MIMIC	6945	1	100%	0.0%	99.8%	100%	99.9%
HP printer Californian	2	2	50.0%	0.0%	40.0%	50.0%	44.7%
Amazon_Dash_Bounty_Button	8	0	100.0%	0.0%	100.0%	100.0%	100.0%
Amazon_Echo Californian	601	38	94.1%	0.2%	73.6%	94.1%	83.1%
Amazon_FireTVStick	138	258	34.8%	0.0%	73.8%	34.8%	50.6%
AMCREST_IPCAM	259	78	76.9%	0.0%	84.6%	76.9%	80.6%
Belkin_Wemo_Switch_Californian	27	18	60.0%	0.0%	84.4%	60.0%	71.1%
D-link_IPCAM	301	30	90.9%	0.0%	90.1%	90.9%	90.5%
FOSCAM_IPCAM	955	80	92.3%	0.0%	96.8%	92.3%	94.4%
FOSCAM_IPCAM_vers2	50	37	57.5%	0.0%	86.2%	57.5%	70.4%
Google_Smartspeaker	146	39	78.9%	0.0%	80.2%	78.9%	79.5%
Philips_Hue	233	28	89.3%	0.0%	95.9%	89.3%	92.5%
RENPHO_humidifier	0	5	0.0%	0.0%	-	0.0%	-
TENVIS_IPCAM	0	1	0.0%	0.0%	-	0.0%	-
TP-Link Smart Plug Californian	34	5	87.2%	0.0%	82.9%	87.2%	85.0%
TP-Link SmartLightBulb	0	4	0.0%	0.0%	0.0%	0.0%	0.0%
Wyze_IPCAM	19	25	43.2%	0.0%	90.5%	43.2%	62.5%
Weighted Avg.	**102380**	**1465**	**98.6%**	**0.8%**	**98.6%**	**98.6%**	**98.1%**

TABLE 9. Combined dataset IoT only: overall accuracy.

Correctly Classified Instances	InCorrectly Classified Instances	Total Number of Instances
102380 - 98.59%	1465 - 1.41%	103845 - 100%

D. HYPER-PARAMETER TUNING

The previous results were obtained using the default settings for training the Random Forest classifier. In this section we explore the impact of changing the maximum allowed depth of the trees. This could be useful for two reasons. First, a larger depth leads to models with higher variance, making the classifier more prone to overfitting. Second, performance may tend to level off with tree depth. In that case, choosing the smallest depth that provides acceptable performance can greatly simplify the evaluation of the model, with lower computational complexity and lower latency.

As in the previous section, the analysis is conducted by splitting the dataset between 70% for training and 30% for test. We analyze the performance of the classifier by performing 5-fold cross validation on the training set (which is, therefore, automatically split between a proper training and a validation set), with tree depth limited to 1, 2, 5, 10, 20, 40 and 80 levels. While we record all the performance metrics, we report here only the MCC, as the other measures follow a similar pattern, and use it to select the optimal depth. The test set is then used to quantify the actual performance of the classifier for the selected depth.

TABLE 10. **Australia IoT with non-IoT traffic.**

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
Dropcam	193	63	75.4%	0.0%	96.0%	75.4%	85.1%
Smart Things	24	8	75.0%	0.0%	100%	75.0%	86.6%
Withings Smart baby monitor	4303	30	99.3%	0.2%	95.1%	99.3%	97.1%
Belkin Wemo Motion Sensor	62395	45	99.9%	0.8%	99.1%	99.9%	99.1%
Samsung SmartCam	5353	15	99.7%	0.0%	99.2%	99.7%	99.4%
Belkin_Wemo_Switch	6247	41	99.3%	0.0%	99.8%	99.3%	99.5%
PIX-STAR Photo frame	794	24	97.1%	0.0%	99.3%	97.1%	98.1%
Amazon_Echo	2907	188	93.9%	0.1%	97.1%	93.9%	95.4%
TP-Link Smart Plug	153	14	91.6%	0.0%	100.0%	91.6%	95.7%
Netatmo weather station	1653	6	99.6%	0.0%	100.0%	99.6%	99.8%
TP-Link DayNight CloudCam	802	49	94.2%	0.0%	99.4%	94.2%	96.8%
Netatmo Welcome	1770	39	97.8%	0.0%	99.7%	97.8%	98.8%
Withings Smart Scale	25	3	89.3%	0.0%	100%	89.3%	94.5%
Triby Speaker	88	49	64.2%	0.0%	91.7%	64.2%	76.7%
NEST Protect smoke alarm	50	17	74.6%	0.0%	100%	74.6%	86.4%
HP printer	52	11	82.5%	0.0%	100.0%	82.5%	90.8%
Insteon Camera	2998	0	100%	0.0%	99.7%	100%	99.9%
Withings AuraSmartSleepSensor	1862	670	73.5%	0.1%	91.2%	73.5%	81.6%
iHome	134	15	89.9%	0.0%	99.3%	89.9%	94.5%
Light Bulbs LiFX SmartBulb	0	29	0.0%	0.0%	-	0.0%	-
Nest Dropcam	243	116	67.7%	0.0%	97.2%	67.7%	81.1%
NON_IoT	45087	595	98.7%	0.9%	98.1%	98.7%	97.6%
Weighted Avg.	**137133**	**2027**	**98.5%**	**0.6%**	**98.5%**	**98.5%**	**98.0%**

TABLE 11. **IoT Australia flows vs. NON_IoT flows, binary classification.**

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
IoT	92750	728	99.2%	1.4%	99.3%	99.2%	97.7%
NON_IoT	45022	660	98.6%	0.8%	98.4%	98.6%	97.7%
Weighted Avg.	**137772**	**1388**	**99.0%**	**1.2%**	**99.0%**	**99.0%**	**97.7%**

TABLE 12. **California IoT with non-IoT traffic.**

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
HP printer	0	4	0.0%	0.0%	-	0.0%	-
Amazon_Dash_Bounty_Button	8	0	100.0%	0.0%	100.0%	100.0%	100.0%
Amazon_Echo	539	100	84.4%	0.3%	80.6%	84.4%	82.2%
Amazon_FireTVStick	84	312	21.2%	0.0%	82.4%	21.2%	41.6%
AMCREST_IPCAM	120	217	35.6%	0.1%	67.8%	35.6%	48.9%
Belkin_Wemo_Switch	21	24	46.7%	0.0%	95.5%	46.7%	66.7%
D-link_IPCAM	255	76	77.0%	0.1%	84.2%	77.0%	80.4%
FOSCAM_IPCAM	421	614	40.7%	0.3%	72.8%	40.7%	53.7%
FOSCAM_IPCAM_vers2	42	45	48.3%	0.0%	72.4%	48.3%	59.1%
Google_Smartspeaker	99	86	53.5%	0.0%	99.0%	53.5%	72.7%
Philips_Hue	231	30	88.5%	0.1%	84.0%	88.5%	86.1%
RENPHO_humidifier	0	5	0.0%	0.0%	-	0.0%	-
TENVIS_IPCAM	0	1	0.0%	0.0%	-	0.0%	-
TP-Link Smart Plug	30	9	76.9%	0.0%	88.2%	76.9%	82.4%
TP-Link SmartLightBulb	1	3	25.0%	0.0%	50.0%	25.0%	35.4%
Wyze_IPCAM	16	28	36.4%	0.0%	94.1%	36.4%	58.5%
NON_IoT	45346	336	99.3%	41.3%	97.0%	99.3%	69.2%
Weighted Avg.	**47213**	**1890**	**96.2%**	**38.4%**	**95.8%**	**96.2%**	**68.9%**

Fig. 9 shows the results for the IoT devices only, for the Australia, the California and the combined datasets, for tree depth up to 40 levels (the values for higher depths do not change relative to 40 levels, and are therefore not shown). The plot includes the results of cross-validation, as well as the results on the test set for all tree depths, where the test set results are significant only after the choice of depth (i.e., they are not used for tuning). In all cases, the performance levels off at a depth of 10 levels, and does not exhibit overfitting. The California dataset, as observed in previous sections, has

TABLE 13. IoT California flows vs. NON_IoT flows, binary classification.

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
IoT	2124	1297	62.1%	0.9%	84.2%	62.1%	70.6%
NON_IoT	45284	398	99.1%	37.9%	97.2%	99.1%	70.6%
Weighted Avg.	**47408**	**1695**	**96.5%**	**35.3%**	**96.3%**	**96.5%**	**70.6%**

TABLE 14. Combined IoT flows with NON_IoT traffic.

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
Dropcam	198	58	77.3%	0.0%	97.5%	77.3%	86.8%
Smart Things	26	6	81.3%	0.0%	100%	81.3%	90.1%
Withings Smart baby monitor	4303	30	99.3%	0.2%	94.9%	99.3%	97.0%
Belkin Wemo Motion Sensor	62392	48	99.9%	0.7%	99.0%	99.9%	99.1%
Samsung SmartCam	5350	18	99.7%	0.0%	99.1%	99.7%	99.4%
Belkin_Wemo_Switch	6248	40	99.4%	0.0%	99.6%	99.4%	99.5%
PIX-STAR Photo frame	792	26	96.8%	0.0%	99.2%	96.8%	98.0%
Amazon_Echo	2905	190	93.9%	0.1%	96.2%	93.9%	94.9%
TP-Link Smart Plug	155	12	92.8%	0.0%	100.0%	92.8%	96.3%
Netatmo weather station	1649	10	99.4%	0.0%	100.0%	99.4%	99.7%
TP-Link DayNight CloudCam	801	50	94.1%	0.0%	98.9%	94.1%	96.5%
Netatmo Welcome	1773	36	98.0%	0.0%	99.6%	98.0%	98.8%
Withings Smart Scale	25	3	89.3%	0.0%	100%	89.3%	94.5%
Triby Speaker	91	46	66.4%	0.0%	91.0%	66.4%	77.7%
NEST Protect smoke alarm	51	16	76.1%	0.0%	98.1%	76.1%	86.4%
HP printer	48	15	76.2%	0.0%	100.0%	76.2%	87.3%
Insteon Camera	2998	0	100%	0.0%	99.7%	100%	99.9%
Withings AuraSmartSleepSensor	1859	673	73.4%	0.1%	90.8%	73.4%	81.4%
iHome	132	17	88.6%	0.0%	99.2%	88.6%	93.8%
Light Bulbs LiFX SmartBulb	0	29	0.0%	0.0%	-	0.0%	-
Nest Dropcam	258	101	71.9%	0.0%	98.1%	71.9%	83.9%
MIMIC	6941	5	99.9%	0.0%	99.9%	99.9%	99.9%
HP printer Californian	1	3	25.0%	0.0%	100.0%	25.0%	50.0%
Amazon_Dash_Bounty_Button	8	0	100.0%	0.0%	100.0%	100.0%	100.0%
Amazon_Echo Californian	540	99	84.5%	0.1%	77.5%	84.5%	80.8%
Amazon_FireTVStick	82	314	20.7%	0.0%	82.0%	20.7%	41.1%
AMCREST_IPCAM	123	214	36.5%	0.0%	67.2%	36.5%	49.4%
Belkin_Wemo_Switch_Californian	21	24	46.7%	0.0%	95.5%	46.7%	66.7%
D-link_IPCAM	259	72	78.2%	0.0%	83.0%	78.2%	80.6%
FOSCAM_IPCAM	413	622	39.9%	0.1%	72.0%	39.9%	53.4%
FOSCAM_IPCAM_vers2	38	49	43.7%	0.0%	67.9%	43.7%	54.4%
Google_Smartspeaker	95	90	51.4%	0.0%	96.9%	51.4%	70.5%
Philips_Hue	225	36	86.2%	0.0%	84.9%	86.2%	85.5%
RENPHO_humidifier	0	5	0.0%	0.0%	-	0.0%	-
TENVIS_IPCAM	0	1	0.0%	0.0%	-	0.0%	-
TP-Link Smart Plug Californian	31	8	79.5%	0.0%	83.8%	79.5%	81.6%
TP-Link SmartLightBulb	0	4	0.0%	0.0%	0.0%	0.0%	0.0%
Wyze_IPCAM	18	26	40.9%	0.0%	94.7%	40.9%	62.2%
NON_IoT	44719	963	97.9%	2.1%	95.4%	97.9%	95.1%
Weighted Avg.	**145568**	**3959**	**97.4%**	**0.9%**	**97.2%**	**97.4%**	**96.5%**

TABLE 15. Combined flows vs. NON_IoT flows, binary classification.

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
IoT	102077	1768	98.3%	2.7%	98.8%	98.3%	95.3%
NON_IoT	44448	1234	97.3%	1.7%	96.2%	97.3%	95.3%
Weighted Avg.	**146525**	**3002**	**98.0%**	**2.4%**	**98.0%**	**98.0%**	**95.3%**

the worst performance, and is considerably lower than the other cases at small tree depths. The performance of the classifier on the test set, here as in the subsequent experiments, does not change significantly relative to the cross validation.

Fig. 10 shows the same data with the inclusion of the non-IoT flows, while Fig. 11 illustrates the performance of the binary IoT/non-IoT classification. The non-IoT flows bring more variety and negatively affect the performance: slightly for the Australia dataset, more significantly for the California

TABLE 16. Combined flows vs. NON_IoT flows, binary classification, 30% test set.

Device	Correct	Wrong	TP rate	FP rate	Precision	Recall	MCC
IoT	30659	530	98.3%	2.8%	98.8%	98.3%	95.2%
NON_IoT	13290	381	97.2%	1.7%	96.2%	97.2%	95.2%
Weighted Avg.	**43949**	**911**	**98.0%**	**2.5%**	**98.0%**	**98.0%**	**95.2%**

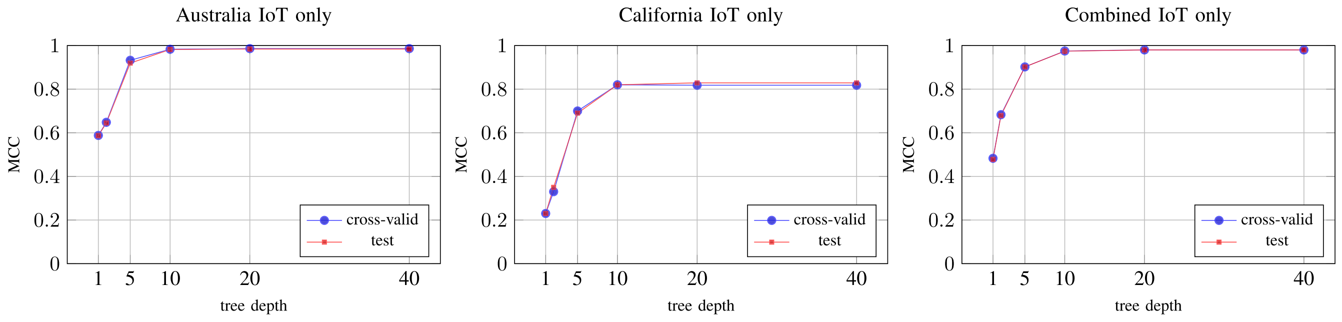

FIGURE 9. MCC of training and test set as the tree depth varies from 1 to 40 for IoT only.

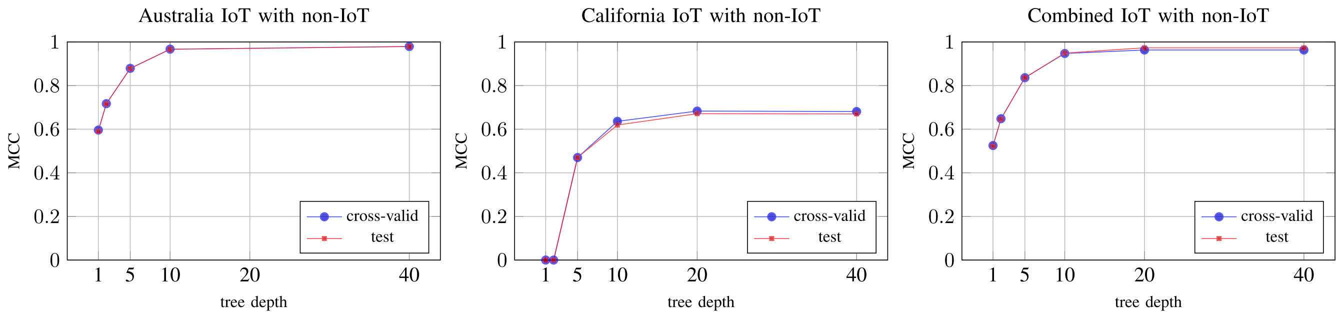

FIGURE 10. MCC of training and test set as the tree depth varies from 1 to 40 for IoT with non-IoT.

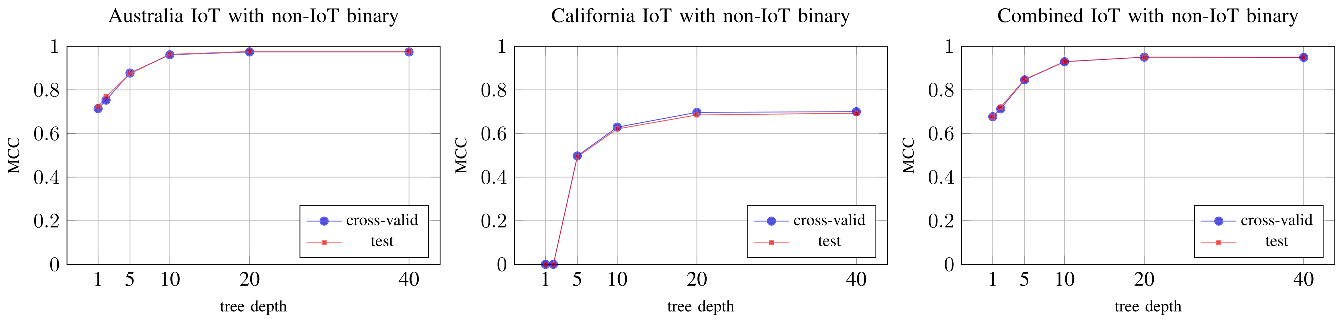

FIGURE 11. MCC of training and test set as the tree depth varies from 1 to 40 for IoT and non-IoT, binary classification.

dataset. In this last case, tree depths less than 5 result in essentially a random classifier. The combined dataset has better performance, largely because of the contribution of the Australia dataset as discussed previously.

It is interesting to validate a classifier derived from one dataset on another dataset. The application is problematic since the devices are not generally the same, but for a few exceptions. We have therefore trained the classifier on the Australia dataset, and validated the model on the California dataset limited to the common devices, i.e., the Belkin Wemo switch, the Amazon Echo and the TP-Link Smart plug, which were given the same labels in the two datasets. The performance in terms of the MCC is shown in Fig. 12.

The results lead to two observations. First, the classifier is sensitive to the specific deployment, and does not perform as well on a different deployment, despite the use of the

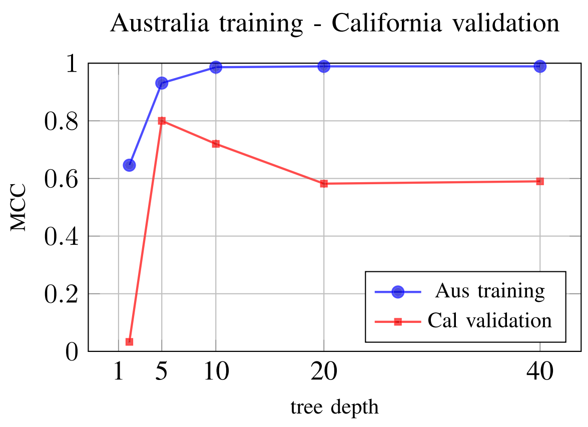

FIGURE 12. Classifier trained on the Australia dataset, validated on the California dataset.

same devices. This may be due to different configurations of the devices, which potentially interact with different servers, altering the dynamics of the communication. More interestingly, we see that the classifier achieves the best generalization at a depth in the range of 5 to 10 levels, with the performance on the validation set decreasing for larger depths. This underscores some amount of variance when the model becomes too complex.

E. DATA AUGMENTATION

We have seen in Section IV-B1 that the recognition rate of devices that have low bandwidth is correspondingly low, to the point that in some cases none of the flows are correctly recognized. This problem can be addressed by using *data augmentation* techniques [46]. Data augmentation consists in enriching the dataset with alternative versions of the data, and is particularly effective in audio and image recognition tasks, where various transforms, such as mirroring or morphing, can be applied to generate additional samples from those already existing in the dataset. Our case is more complicated, as an arbitrarily altered flow most likely no longer belongs to the original class nor to the original protocol, and could in fact reduce the quality of the dataset. Knowledge of the protocol format and of the application can lead to crafted flows that are valid, for instance through the use of simulation tools [31], such as the Mimic simulator we have used in our dataset. Unlike image classification, where data augmentation is used to add variety for better generalization, the problem we face in our case is that certain classes are significantly under-represented in the dataset. The learning process, therefore, struggles to adjust the weights of the inference algorithm to properly account for them. We therefore evaluate a simpler approach to data augmentation, where flows present in low numbers and with low classification performance are simply replicated multiple times to gain additional weight in the dataset. While this does not provide new information to the model, it does help to achieve better balance during the optimization.

For simplicity, we illustrate the procedure on the California dataset only, which has a lower classification performance than the Australia dataset according to the results shown in Section IV-B1 and IV-B2. In the following, for all cases, we split the data into a 70% training set and a 30% test set, as explained in Section IV-C, used for validation. To reduce the size of the labels in the graphs, we denote the individual devices as shown in Table 17.

TABLE 17. Device abbreviations for the California dataset in the graphs.

Device	Abbr.	Device	Abbr.
HP printer	HP	Amazon Dash Bounty Button	ABB
Amazon Echo	AE	Amazon FireTVStick	ATV
AMCREST IPCAM	AIP	Belkin Wemo Switch	BW
D-link IPCAM	DIP	FOSCAM IPCAM	FIP
FOSCAM IPCAM vers2	FIP2	Google Smartspeaker	GSS
Philips Hue	PH	RENPHO humidifier	RH
TENVIS IPCAM	TIP	TP-Link Smart Plug	TPP
TP-Link SmartLightBulb	TSB	Wyze IPCAM	WIP
NON IoT	NIOT		

We then proceed as follows. We consider the device with the lowest or undefined value of the MCC as the candidate for augmentation, and replicate its flows in the dataset multiple times. Then we shuffle, split, retrain and validate. We start with the Tenvis IPcam (label TIP), which has only one flow in the dataset. This is indeed an extreme case, but nevertheless a potential outcome of data collection in practice. Fig. 13 shows the results of replicating the Tenvis IPcam once (i.e., no replication), 10, 20, 40 and 80 times.

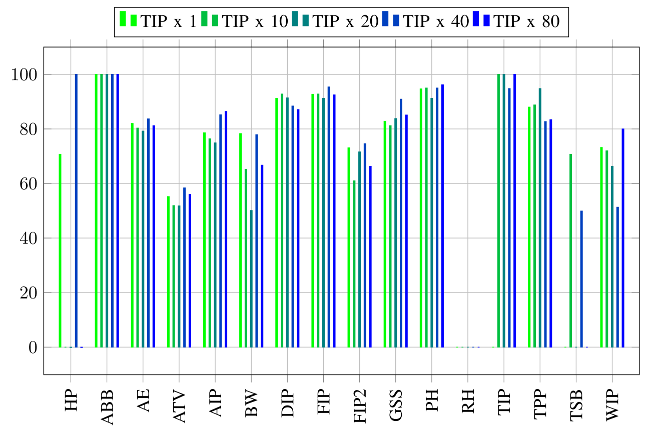

FIGURE 13. Variation of MCC with increasing replication of TIP flows.

As we replicate the instances, the MCC of the chosen device rapidly increases to 100%, already with 10 flows. In the study, we have further replicated to observe the impact on the classification of the other flows. We see that while some devices remain roughly unchanged, other have significant swings. We therefore proceed to replicate additional devices. Fig. 14 shows the change in MCC as we replicate the Renpho Humidifier (label RH) by the same amounts, keeping TIP replicated 80 times. Again we observe a rapid increase in performance for RH.

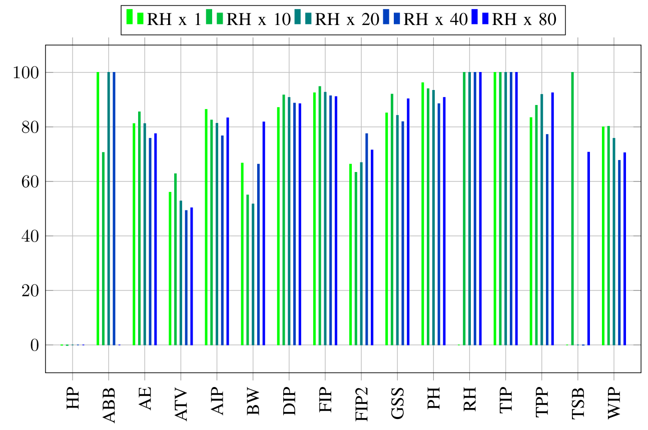

FIGURE 14. Variation of MCC with increasing replication of RH flows.

We continue with the same methodology, and replicate HP Printer (HP) first and the TP-Link SmartLightBulb (TSB) next by the same amounts, keeping the previous replications. Fig. 15 and Fig. 16 show the corresponding variations.

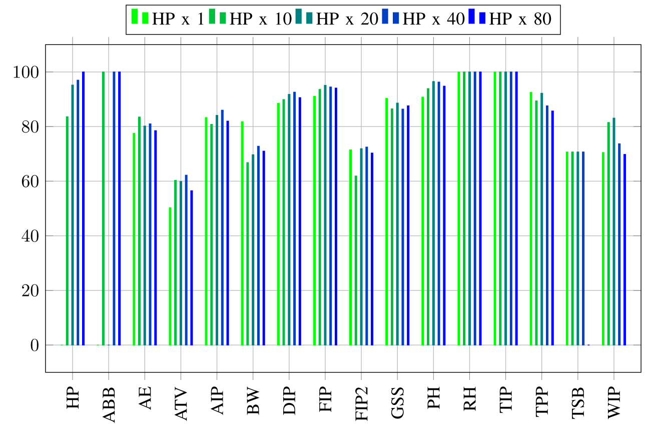

FIGURE 15. Variation of MCC with increasing replication of HP flows.

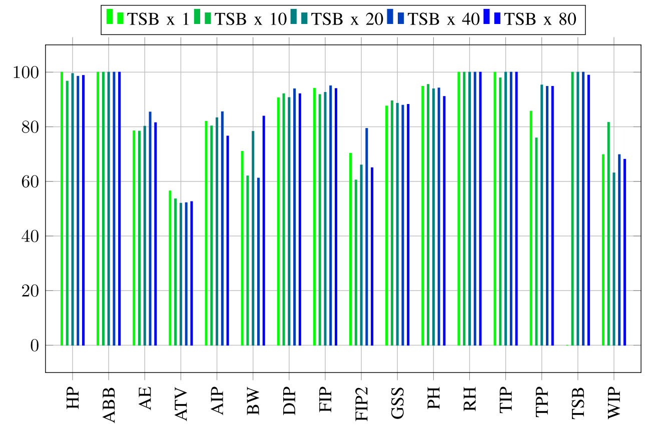

FIGURE 16. Variation of MCC with increasing replication of TSB flows.

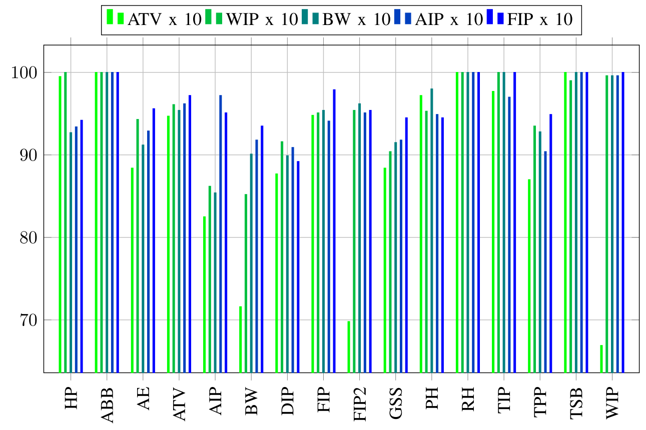

FIGURE 17. Variation of MCC with increasing replication of flows from ATV, WIP, BW, AIP, and FIP.

The results show that we have now reached satisfactory results in the under-represented categories. To further improve classification performance, we replicate in sequence the flows of the least performing devices, this time only by 10 times to avoid breaking the balance that we have achieved so far. Fig. 16 shows the MCC as we replicate the flows for ATV, WIP, BW, AIP, and FIP.

Finally, we put back the non-IoT flows in the picture, which tend to decrease the classification performance on the devices. We therefore further duplicate DIP, ABB, TPP, PH and AE in sequence to regain accuracy. These final values are shown in Fig. 18.

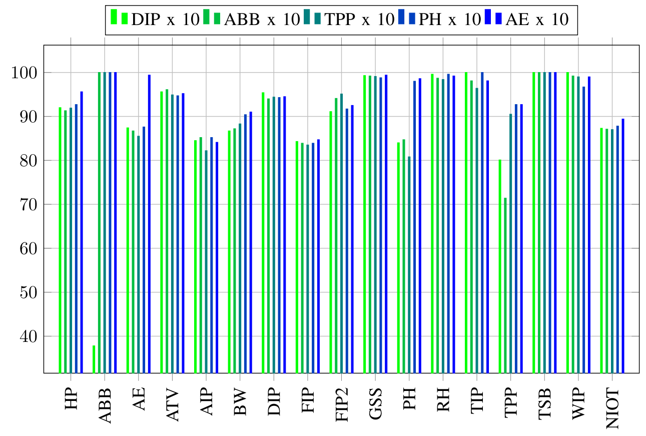

FIGURE 18. Variation of MCC with replication of DIP, ABB, TPP, PH and AE, including non-IoT flows.

The results that we have achieved show that this simple technique could be useful in balancing the distribution of the dataset, therefore letting the learning algorithm better converge to an optimal point. The final weighted average of the MCC on all categories including the non-IoT flow (the blue bars in Fig. 18) reaches 90.7%, a significant improvement over the 68.9% obtained with the original data (see Table 12).

F. DYNAMIC TRAINING

We have seen that classification is generally poor when a device that was not present in the training set, or that has a materially different configuration, is added to the mix of the test set. When the classifier is used to process packet flows in real time, we would like the system to automatically detect this situation, and raise a flag so that the operator can take appropriate recovery actions, in a spirit similar to the dynamic learning approach proposed by Grimaudo *et al.* [47] in the context of unsupervised learning. These actions include, for instance, an inspection of the logs and a potential retraining of the classifier with new data. In this section we explore how to implement this process.

To properly run the random forest implementation we move from Weka, which is convenient to quickly visualize and train multiple models, to the Scikit-Learn framework [36], which provides an easier environment for runtime experimentation in Python. The datasets remain unchanged. The essential feature that we take advantage of is the additional information on the quality of the prediction that we obtain from the outputs of the individual classification trees in the forest. In particular, each tree provides its estimated probabilities of classification for all available classes. An average is then taken over all the trees, and the classification result corresponds to the class with the largest probability value. By inspecting this value we can determine if the classification had a high or low confidence.

The individual values of the probabilities actually fluctuate even for the flows that are present in the training set, and which have generally high classification confidence. For this reason, it is impractical to flag every individual flow for which confidence is low. Instead, we proceed by calculating an exponentially weighted average of the classification probability over the past flows. To do so, we compute the current confidence level as the combination of the confidence level computed for the previous flow, and the classification probability of the current flow, according to:

$$c = \beta c + (1 - \beta)p, \tag{1}$$

where c is the confidence level, while p the probability of the latest classification. By changing the value of β, we can modulate the effective number of previous flows which are taken into consideration, as their weight decreases exponentially. For instance, a value $\beta = 0.9$ accounts for roughly the previous 10 flows, while $\beta = 0.99$ for the previous 100 flows. To maintain a sufficiently smooth variation, we experiment with 100, 1,000 and 10,000 previous flows.

The strategy that we adopt works as follows. We train the classifier with an initial set of devices, and then test its performance by evaluating one flow at a time of the corresponding test set, while keeping track of the confidence level c. At some point during the test, we mix new devices on which the classifier was not trained in the test, and observe the confidence decrease. If the confidence crosses a threshold, then we stop the evaluation, create a new training and test set with both the old and the new devices, retrain the classifier, and resume the test. The confidence level will then increase and reach a new steady state level. We repeat the process with yet new devices to continue the evaluation.

More specifically, we illustrate the approach with a large set that comprises devices from both the Australia and the California dataset, and divide it into 7 subsets S_i containing devices with decreasing classification accuracy. Each subset is further divided into a 70% training set S_i^t, and a 30% test set S_i^v, as in the previous sections. We then train the classifier with S_1^t and start feeding it the flows in the test set S_1^v, while keeping track of the confidence level. Once we are through two thirds (2/3) of the test set, we mix in random flows from the second test set S_2^v, which contains currently unknown devices, to get around half old and half new flows. If the confidence decreases below a 0.9 threshold, we take the union of S_1^t and S_2^t, and retrain the classifier. We then continue the test, and repeat the procedure with S_3 once we are through two thirds of the test set. The whole process continues with the remaining subsets.

The results for $\beta = 0.999$ (which accounts for approximately the previous 1,000 flows) are shown in Fig. 19, where the confidence level is plotted against the number of flows. The vertical red lines denote the points where new devices are introduced to the test. Initially confidence is high. Then, at flow 14,519 we introduce the new devices and confidence decreases down to the threshold. We then retrain, and start testing again, and see the confidence approach the initial level again, as expected. We repeat the procedure with the third test S_3^v set at flow 32,323. This set comprises fewer flows, and while confidence decreases it does not reach the threshold. We therefore mix in the test flows from the fourth dataset S_4^v, reach the threshold and retrain with the union of the training subsets. For the fifth dataset, at flow 50,987, we start testing with the new flows in proportion of 50%, then decrease their presence in the mix to a low level. Confidence initially decreases, then increases as classification becomes generally more accurate, however it is slightly more noisy. At flow 70,672 we mix in S_6^v and subsequently S_7^v, reach the threshold and retrain. Overall, confidence has a decreasing trend, because of our choice of progressively including devices with lower classification accuracy.

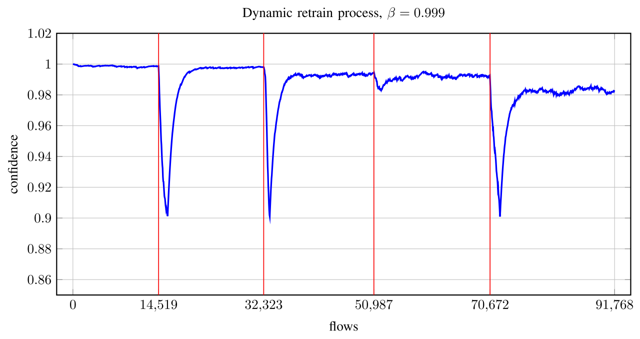

FIGURE 19. Confidence level as a function of time as flows of new devices are introduced to the mix, $\beta = 0.999$.

The value of β can be chosen to adjust the inertia of the system. For illustration, Fig. 20 shows the same experiment with $\beta = 0.99$, i.e., by mostly considering only the previous 100 flows. The system responds much more quickly to changes in the test set, however the curve is considerably more noisy, and could, in certain circumstances, raise too many flags. Experiments with $\beta = 0.9999$ (for approximately 10,000 previous flows) indicate that the system responds too slowly for our set up.

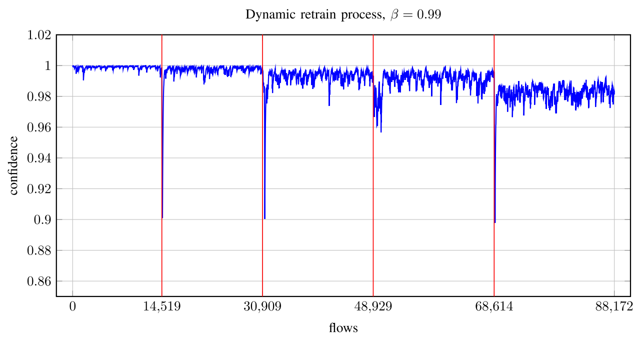

FIGURE 20. Confidence level as a function of time as flows of new devices are introduced to the mix, $\beta = 0.99$.

G. IMPLEMENTATION AND PERFORMANCE

In this section we explore the actual real-time performance of our approach. The Python SciKit-Learn implementation of the prediction function is optimized using vector operations to handle large numbers of test examples in a single function call. This is not suitable for a deployment where flows must be processed for prediction one at a time as they are received. Indeed, experiments show that the overhead due to the interpreted nature of Python makes this solution impractical, as the prediction may take milliseconds to be performed. For this reason, we have turned to a native C implementation to assess the runtime performance. We consider two different approaches. In the first, the trees are translated into a nested structure of if-then-else statements, where leaves return the classification results, while internal nodes switch branches according to the selected features and their values. We refer to this solution as the *if-then* algorithm. The second implementation encodes the trees into an array, where each element is a structure that represents either a leaf with the classification, or an internal node with its branching information. A simple algorithm is used to traverse the array and reach a classification. This solution is denoted as the *struct* algorithm. In both cases, the C code is automatically generated from the trained model by recursively analyzing the internal SciKit-Learn structure. We consider and compare two versions for each algorithm: one trained with *unconstrained tree depth*, and one where the tree depth is *limited to at most 10 levels*, according to the accuracy results reported in Section IV-D. In the unconstrained case, the generated trees have a depth that ranges between 19 and 28 levels, with an average of 23.08. The number of trees is fixed to 100 in all cases. In addition, we will consider both a *single-threaded sequential* implementation and a *multi-threaded parallel* execution.

The code for the trees is complemented by i) the FFT algorithm, ii) a routine to compute the magnitude of the FFT, and iii) a decision algorithm that computes the final classification and the confidence level on the basis of the result of the individual trees. For the FFT we have resorted to the optimized FFTW v3.3.8 library, compiled with support for the processor SIMD extensions such as AVX2, and configured to use single floating point precision. The magnitude of the FFT is instead computed using the hand-written routine of the Vector-Optimized Library of Kernels (VOLK),[10] which also takes advantage of the SIMD extensions. The decision algorithm, on the other hand, simply inspects the results of the individual trees to take the majority voting and updates the confidence level according to Eq. 1.

The code is compiled with gcc version 5.4.0, with all optimizations enabled (-O8). All experiments are conducted on a desktop PC running Linux Ubuntu 16.04, equipped with a 4-core, 8-thread Intel Core i7-6700 CPU clocked at 3.40 GHz, with 8 GByte of RAM. Performance is measured by instrumenting the code with statements that read the clock value at intermediate points with nanosecond accuracy, to isolate the contribution of the FFT, the magnitude computation, the tree traversal, and the decision process, as well as to provide an overall per flow measurement. We process the entire set of flows used in the previous section on dynamic training, totaling 95,837 flows, one at a time, in the form of arrays of packet lengths, repeating the whole procedure from first to last flow 100 times, for a total of nearly 10 million flows. The sequence of flows is randomized. Indeed, if flows of the same kind are grouped together and processed sequentially, performance increases artificially (by almost a factor of 2 in our experiments) because the cache and especially branch prediction work under ideal conditions. The performance data is stored in arrays as flows are inspected, and later processed to compute the average, the standard deviation, and to generate the runtime and the cumulative distributions that we show next. The distributions are computed by counting the number of executions for every nanosecond, and then taking a 10 ns moving average to smoothen out random variations.

1) SEQUENTIAL IMPLEMENTATION, *if-then* ALGORITHM

In our first set of experiments, flows are classified sequentially by the *if-then* algorithm. Table 18.a summarizes the results for the unconstrained depth version, while Fig. 21 shows, on a linear scale, the corresponding runtime distributions, for the individual steps of the computation, and for the overall classification procedure. We observe that the magnitude computation, the decision algorithm and to a lesser extent the FFT, are quite deterministic, as witnessed by the well defined peak in their runtime distribution, and the low standard deviation. We remark that to compute the standard

[10]http://libvolk.org/

TABLE 18. **Overall running times of flow classification, *if-then* algorithm.**

a) *if-then*, unconstrained depth

Phase	average	std. dev.	min	max
FFT	254 ns	59 ns	183 ns	1,603 ns
Magnitude	57 ns	10 ns	45 ns	1,391 ns
Tree	2,166 ns	2,690 ns	252 ns	44,410 ns
Decision	156 ns	8 ns	65 ns	1,439 ns
All	2,622 ns	2,697 ns	639 ns	45,794 ns

b) *if-then*, depth = 10

Phase	average	std. dev.	min	max
FFT	246 ns	45 ns	181 ns	1,474 ns
Magnitude	57 ns	10 ns	45 ns	1,379 ns
Tree	1,364 ns	1,243 ns	356 ns	28,162 ns
Decision	151 ns	12 ns	64 ns	1,417 ns
All	1,809 ns	1,257 ns	743 ns	28,512 ns

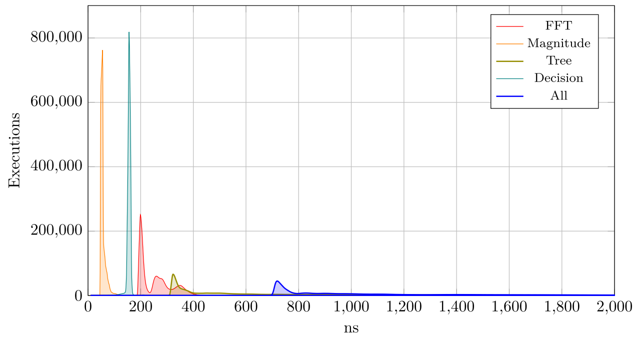

FIGURE 21. **Runtime distributions of FFT, magnitude computation, tree traversal, decision algorithm, and overall classification. Unconstrained depth. Linear scale.**

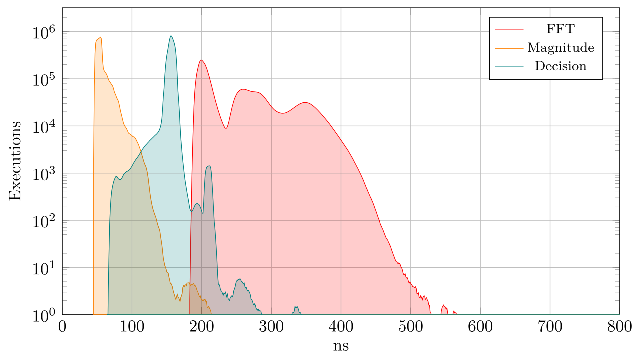

FIGURE 22. **Runtime distributions of FFT, magnitude computation and decision algorithm. Unconstrained depth. Logarithmic scale.**

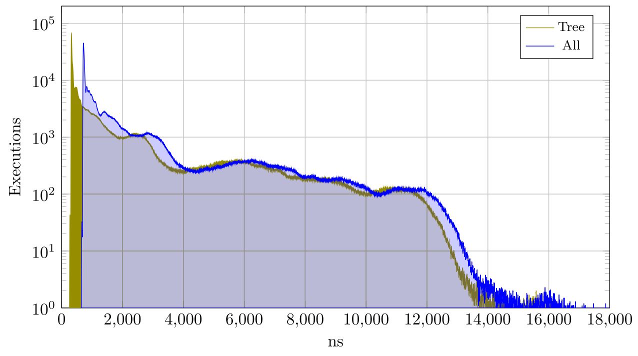

FIGURE 23. **Runtime distributions of tree traversal and overall classification. Unconstrained depth. Logarithmic scale.**

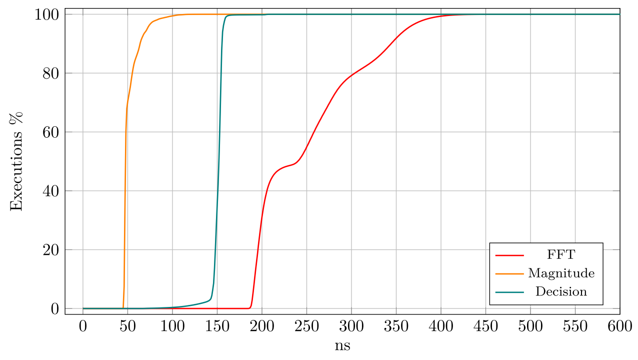

FIGURE 24. **Cumulative distribution function of the runtimes of the FFT, magnitude computation and decision algorithm. Unconstrained depth.**

deviation we have left out a few outliers (with running times of more than 2 μs), which total less than 0.01% of the flows, due to non-deterministic scheduling effects. These outliers essentially do not affect the average, but have a material impact on the standard deviation. Conversely, the tree traversal, besides having a larger average runtime, also has a considerable standard deviation. This is not due to scheduling, but is the effect of traversing different portions of the trees on account of the different packet size distributions of each flow. As a consequence, its distribution has a much lower peak and a long tail. The same is true for the overall classification, which is essentially a translated version of the tree traversal that includes the contribution from the computation of the other steps.

Fig. 22 and Fig. 23 show the same information on a logarithmic scale, and on different ranges of the execution times, to highlight the details of the runtime distributions. In particular, Fig. 23 shows the tree traversal long tail, which dies out at around 16 μs. One must take this behavior into account if the interest lies in the latency of the classification, rather than the throughput. To complement the runtime distributions, Fig. 24 and Fig. 25 show the cumulative distribution functions for the FFT, the magnitude computation and decision, and for the tree traversal and overall classification, respectively. In particular, from Fig. 25 we observe that 80% of the flows are classified within 3.8 μs, 90% within 6.8 μs, 95% within 8.9 μs, and 99% within 11.8 μs. On average, the algorithm executes 381 thousand classifications per second.

When the model is limited to a tree depth of just 10 levels, the performance of the tree traversal is much more deterministic, as well as considerably faster. Table 18.b shows the results. Whereas the FFT, the magnitude computation and the decision are essentially unchanged, the tree traversal is on average 1.6 times faster, with less than half the standard

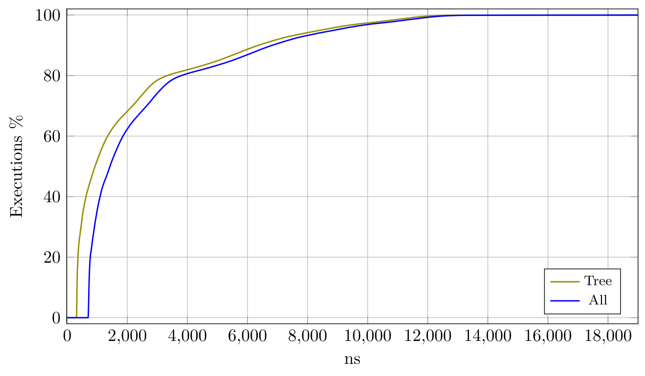

FIGURE 25. Cumulative distribution function of the runtimes of tree traversal and overall classification. Unconstrained depth.

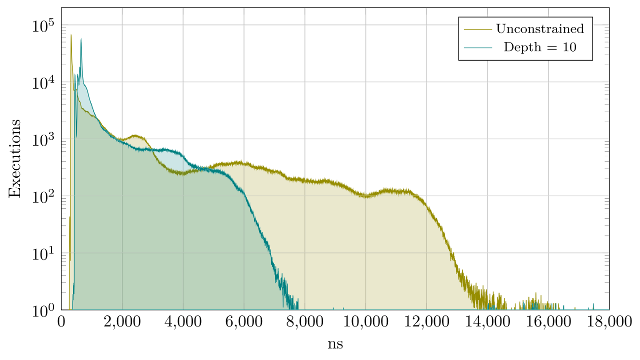

FIGURE 26. Comparison of runtime distributions for tree traversal, unconstrained depth vs. depth limited to 10 levels.

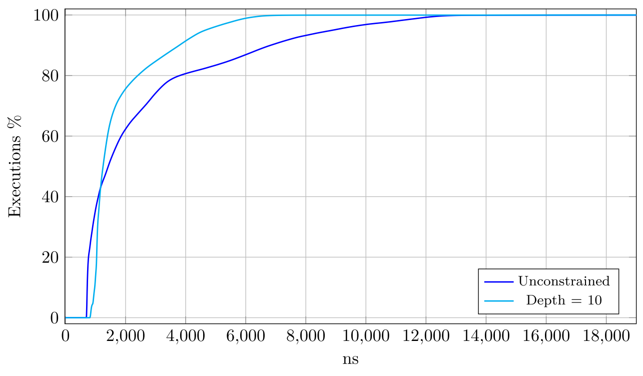

FIGURE 27. Cumulative distribution functions, overall classification, comparison between unconstrained depth vs. depth limited to 10 levels.

TABLE 19. Overall running times of flow classification, *struct* algorithm.

a) *struct*, unconstrained depth

Phase	average	std. dev.	min	max
FFT	253 ns	12 ns	205 ns	1,547 ns
Magnitude	48 ns	5 ns	45 ns	747 ns
Tree	3,289 ns	3,386 ns	976 ns	51,794 ns
Decision	26 ns	4 ns	21 ns	1,419 ns
All	3,618 ns	3,387 ns	1,286 ns	52,120 ns

b) *struct*, depth = 10

Phase	average	std. dev.	min	max
FFT	257 ns	7 ns	198 ns	1,418 ns
Magnitude	48 ns	4 ns	45 ns	1,001 ns
Tree	2,508 ns	1,756 ns	938 ns	45,974 ns
Decision	26 ns	4 ns	23 ns	1,175 ns
All	2,841 ns	1,761 ns	1,213 ns	46,304 ns

deviation. This, of course, is also reflected on the overall performance of the classification. Fig. 26 compares the runtime distributions of the two versions for the tree traversal only, highlighting the much shorter tail when the depth of the tree is limited to 10 levels only. Likewise, the cumulative distribution function for the overall classification, shown in Fig. 27, underlies that the majority of the flows are classified within 6 μs. Notice how the constrained depth means that more flows require a deeper traversal of the tree, shifting the cumulative distribution slightly to the right relative to the unconstrained case. On average, the algorithm executes 553 thousand classifications per second.

2) SEQUENTIAL IMPLEMENTATION, *struct* ALGORITHM

The *if-then* algorithm is a straightforward implementation that reduces overhead, but may be inconvenient when re-training the model, as the executable must be replaced with a new one. Encoding the tree into a data structure and traversing it, as done in the *struct* algorithm, results in simpler management, as processes need not be terminated and only data must be updated. Another advantage is that one can switch among different models by simply changing the value of a pointer to link to different tree structures. This flexibility is balanced by a slight decrease in performance, as shown in Table 19. We observe that the *struct* algorithm is 1.38 times slower than the *if-then* implementation in the unconstrained depth case, and 1.57 times slower when depth is limited to 10 levels. Notice that a portion of the decision (incrementing of the class counts) was shifted directly into the tree traversal, resulting in lower execution time for the decision portion.

Fig. 28 and Fig. 29 compare the distribution of the runtime of the tree traversal for the *if-then* and the *struct* algorithm in the two cases of unconstrained depth, and depth limited to 10 levels, respectively. The diagrams show both a longer tail for the *struct* algorithm, as well as a higher minimum traversal time for the fastest flows.

3) MULTI-TASKING EFFECTS

If we look closely at Fig. 29, we see a number of executions with runtimes between 14 μs and 20 μs. These are outliers which are in fact common to all phases of the classification algorithm (including the FFT, magnitude and decision) and are attributable to random effects of the operating system multi-tasking scheduling algorithm. This can be confirmed by inspecting the runtime for the same flow across the 100 executions of the experiment. In general the runtimes for the same flow have low standard deviation, with occasional long executions, which change position if the experiment

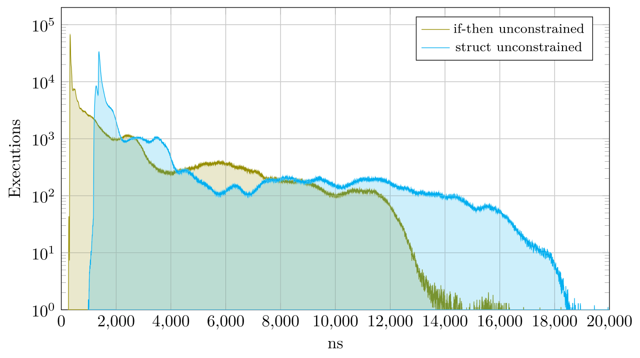

FIGURE 28. Comparison of runtime distributions for tree traversal, *if-then* vs. *struct* algorithm, unconstrained depth.

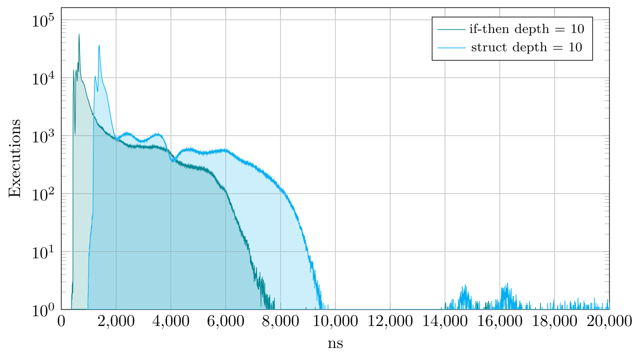

FIGURE 29. Comparison of runtime distributions for tree traversal, *if-then* vs. *struct* algorithm, depth limited to 10 levels.

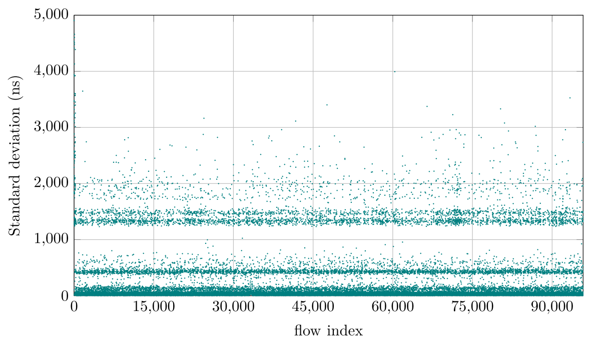

FIGURE 30. Standard deviation for each flow during tree traversal, *if-then* algorithm, unconstrained depth.

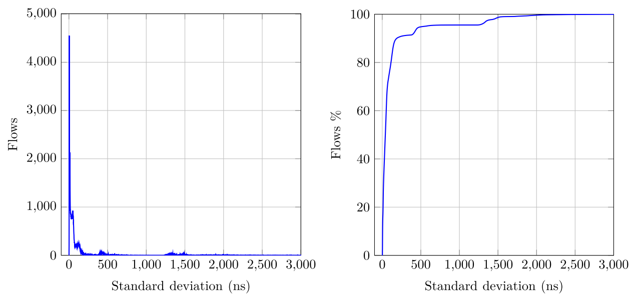

FIGURE 31. Distribution and cumulative distribution of the standard deviation for each flow during tree traversal, *if-then* algorithm, unconstrained depth.

is repeated different times. Consider the *if-then* algorithm with unconstrained depth. Fig. 30 shows for each flow the standard deviation of the tree traversal runtime across the 100 executions, while Fig. 31 shows their distribution and cumulative distribution. The vast majority of the flows have very low standard deviation, with an average of 132 ns (much lower than the standard deviation across all flows and all executions), indicating that the classification of each kind of flow is generally rather deterministic. The horizontal bands in Fig. 30, on the other hand, suggest that the scheduling mechanisms of the operating system affect the execution times, according to defined periodicities. Dedicated servers should therefore expose controls over these mechanisms, in order to reduce their impact and increase determinism.

4) PARALLEL IMPLEMENTATION

The classification of each flow is independent of the classification of any other flow. Consequently, the classification algorithm can be run in parallel for different flows in a multi-threaded implementation, as suggested by Van Essen *et al.* [48] (see Section IV-H for a comparison). This way, a new classification can start as new flows are received, even if the previous classification has not yet terminated. Conceivably, one could even run different *trees* in parallel for the same flow. However, because individual trees execute quickly, the overhead due to thread synchronization tends to cancel the advantage of the parallel execution. We therefore limit ourselves to parallelizing the flows.

Our implementation is based on the pthread library [49]. A configurable number of threads are initially created to handle equal portions of the flows, they are started, and independently run to sequentially classify their assigned flows. The main thread then waits until all threads have completed their executions. In these experiments we are unable to keep track of the execution times of the inner parts of the algorithm, as executions overlap and would not give a proper measure. Instead, we measure the total execution time, and compute the average per-flow running time. For illustration, we consider the *if-then* algorithm, in the two variants of unconstrained depth and depth limited to 10 levels, as the number of threads is increased. Since the processor supports up to 8 independent threads, we expect the speed-up to level off after this value.

The results are shown in Fig. 32 for both the unconstrained depth and the 10-level limit case, as the number of threads varies from 1 to 16. As expected, the execution time decreases quickly up to 4 threads, corresponding to the number of cores of the platform we are using. Performance further improves up to the 8 threads supported by the processor, however threads must compete for resources, and gains are therefore more limited. No further gains can be achieved with more threads, in fact there is a slight amount of overhead that decreases performance. Table 20 summarizes the performance for the 2-, 4- and 8-thread case. Compared to the

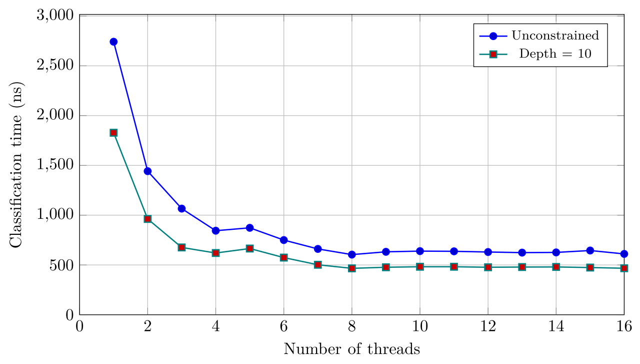

FIGURE 32. Average flow classification time as the number of parallel threads is increased from 1 to 16, *if-then* algorithm.

TABLE 20. Per-flow execution time, *if-then* algorithm, by number of threads.

a) *if-then*, unconstrained depth

Threads	average	std. dev.	min	max	sp.-up
2	1,442 ns	12 ns	1,433 ns	1,483 ns	1.82
4	844 ns	107 ns	781 ns	1,082 ns	3.11
8	604 ns	36 ns	578 ns	790 ns	4.34

b) *if-then*, depth = 10

Threads	average	std. dev.	min	max	sp.-up
2	962 ns	7 ns	955 ns	999 ns	1.88
4	620 ns	126 ns	530 ns	829 ns	2.92
8	466 ns	25 ns	446 ns	591 ns	3.88

sequential implementation, the parallel execution achieves a maximum speed-up of 4.34 times in the unconstrained depth case, and of 3.88 times when the depth is constrained to 10 levels, with the bulk of the gains reached already at 4 threads. Overall, the parallel versions are able to execute 1.66 million classifications per second and 2.15 million classifications per second, respectively. The results show that the algorithm can effectively be parallelized, even though gains are eventually bounded by the competition of the threads on the processor resources.

H. DISCUSSION

The classification results shown in the previous sections highlight that the analysis of the series of the payload lengths in the frequency domain provides high selectivity, with high overall accuracy and MCC. On the other hand, the analysis also shows that smaller datasets may be unable to provide sufficient training examples to reduce the classification error. In this case, the model may tend to overfit part of the data, and the validation suffers. Data augmentation can help reduce this effect. Our analysis shows that balancing the number of flows in the dataset, by duplicating the lower bandwidth devices, provides better performance and helps the learning algorithm reach higher classification accuracy. Another possible remedy in these cases is to further reduce the number of features. In a preliminary experiment, we have considered a reduced subset of the Australia dataset corresponding to only three days of data capture with approximately 17,000 flows. We have then isolated only a reduced number of peaks in the spectrum. The experiments show that using only 32, 16 or even 4 peaks reduces accuracy by only 0.2% points, showing that the highest values of the spectrum carry the majority of the information. A more detailed evaluation of this approach is part of our current and future work.

The classification accuracy and performance in terms of precision, recall and the other metrics that we obtain is in line with the results obtained in our previous work [12], however we here use a very different set of features. More specifically, we focus on simplified features based on the length of the packets (in some cases complemented by the inter-arrival times) and further evaluate the performance on a much larger dataset. This is convenient for both computational complexity, if the recognition must be run in real time, and to handle flows that are obfuscated by encryption. A combined classifier could also be used to increase the reliability. In addition, we have extended the classifier to distinguish between the different classes of devices, a valuable information to adjust the network quality of service, to detect anomalous behavior and therefore isolate compromised devices.

Other methods, discussed in Section II, have also been presented in the literature that reach high classification accuracy, selectivity and specificity [15], [16], [25], [26]. Our intention was not so much improving the performance metrics, which arguably reach almost perfect classification in some of the reported work, but rather achieve similar performance using a considerably reduced set of features that does not need header or payload inspection. By doing so, the classifier could be deployed also in the presence of encryption. In addition, we study the performance of classification in the frequency domain.

In our study we also explore the impact of tree depth in terms of classification performance. The results show that relatively shallow trees already provide the bulk of the distinguishing power of the method, while resulting in a much simpler implementation, essential when dealing with traffic in real time. We have also considered applying the classifier obtained with one dataset to an entirely different dataset. This, as far as we know, has never been attempted in the literature. The analysis, limited to the common devices, shows that i) performance suffers, and ii) deep trees may overfit relative to another dataset. The tree depth analysis identifies the optimal value for the hyper-parameter in this case. Nevertheless, the models show some difficulty in generalizing, presumably because of the wide differences in configurations and environment of operation. This aspect is largely unexplored in the state of the art, and is hampered by the lack of appropriate labeled datasets. One mention is given by Pinheiro *et al.* [16], who observe similar behavior with firmware updates or compromised devices, and suggest, without going further, that unsupervised learning techniques could be used to address the problem.

Real-time classifiers, exposed to a potentially changing mix of different devices in the network, must be able to detect when confidence in classification is poor. We have implemented this feature by taking an exponentially weighted average of the classification probability output by the trees over the past flows. A guided retraining phase can be initiated whenever the confidence level decreases below a set threshold.

We have extensively characterized the real-time performance of our classifiers, analyzing the contribution of the different phases of the algorithm, and considering different implementations. Table 21 summarizes the results in terms of kilo-classifications per second (kcps). To put these numbers in context, a 10 Gbps Ethernet network, with a minimum packet size of 84 bytes, can carry at most 14.9 Mpps (million packets per second). Our best algorithm could therefore run on a network of this kind, in the conservative assumptions that flows have at least 7 packets on average. In general, packets are larger than minimum size and flows are longer (however this depends on the specific application), making our approach suitable for high speed networks. As mentioned earlier, in our dataset IoT devices have an average of 20 packets per flow, and 365 bytes per packet. Thus, on average, we need to run a classification every 7,300 bytes. For a 100 Gbps network, this translates into 1,712 kcps, well within the performance of the multi-threaded depth-limited implementation, and within reach of the multi-threaded unconstrained implementation. Introducing non-IoT flows in the mix, which are longer and larger (146 packets per flow, 560 bytes per packet in our dataset), can only improve the performance.

TABLE 21. Real-time performance, classifications per second.

Algorithm	Class. per sec.
if-then unconstrained depth	381 kcps
if-then depth = 10	553 kcps
struct unconstrained depth	276 kcps
struct depth = 10	352 kcps
parallel unconstrained depth	1,656 kcps
parallel depth = 10	2,146 kcps

Several further optimizations have been reported in the literature, essentially using parallelism in various forms. Van Essen *et al.* study the scalability of random forest on multi-threaded CPUs, on general purpose GPUs and on FPGAs [48]. The CPU implementation is similar to our *struct* algorithm, and is parallelized using openMP on a 2-socket Intel X5660 Westmere system with 12 cores running at 2.8 GHz with 96 GByte of DRAM. With 32 trees and a depth of 6 levels, they achieve 9,291 kcps with 12 threads, and 884 kcps with 1 thread (considering the tree execution time only). With 234 trees and again a depth of 6 levels, the performance is 1,044 kcps with 12 threads and 93 kcps with 1 thread. Our results are generally in line, or somewhat better when scaled for the number of trees and depth, although a lot depends on the complexity of the classification. The authors do not report the standard deviation of their executions. What we can extrapolate is instead the performance improvement due to the use of GPUs (a Tesla M2050) and FPGAs (a Xilinx XC6VLX240T-1), which run 2 to 5 times faster than the multi-threaded CPU, with a much more favorable performance/power trade off.

In addition to GPUs and FPGAs, our algorithm could easily be implemented on network processors that offer increasing levels of multi-threaded performance. For instance, the Mellanox Indigo NPS-400 network processor [50] supports up to 4,000 threads on 256 cores, with a modified version of the Linux operating system that eliminates the scheduling overhead, mitigating the effects we have described in Section IV-G3 and simplifying synchronization. With dedicated hardware for flow management, performance on this platform could easily reach network speeds well in excess of 100 Gbps.

V. CONCLUSION

In this paper, we have studied a statistical classification method to discriminate between IoT and non-IoT traffic, and to determine the device that originates the communication flow. We have first presented and characterized our dataset, collected from repositories made available in the public domain, discussed the tools we have used to capture the data, generate the flows and their statistics, and construct the classification algorithms. We have shown the features of different classes of devices, and discussed the classification performance of the Random Forest algorithm using 10-fold cross validation. We have then analyzed the impact of tree depth to the performance metrics and selected the optimal value to trade off accuracy and computational complexity. When devices have low bandwidth, the relative weight of their flows in the dataset is correspondingly low. We show that replication can be used to balance the dataset, and direct the learning algorithm to a better overall optimal point. We have used a confidence metric to estimate the accuracy of the classifier, and detect when a new training phase is necessary due to the introduction of previously unknown devices. We have illustrated this technique by implementing the classifier in Python using SciKit-Learn. Finally, we thoroughly analyze the runtime performance of the classifier, and show that a native multi-threaded C implementation automatically generated from the trees is able to handle very high speed networks.

In our future work, we plan explore the timing relation among different flows attributed to the same device. The difficulty with this kind of analysis is that regularity is seen across the flows which are opened and closed by a particular device. A different criterion must therefore be devised. In the context of cellular networks, one can use a specific identifiers, like the IMSI, to associate traffic to a device. At the same time, the same device, such as a smartphone, might behave as both a "thing" (by using its sensors) and a traditional terminal, creating confusion or noise in the classification. Regarding the dynamic learning technique, additional information, such as the port number (as used in [47]), could also be inspected to estimate accuracy, when the information is available.

REFERENCES

[1] S. B. Baker, W. Xiang, and I. Atkinson, "Internet of Things for smart healthcare: Technologies, challenges, and opportunities," *IEEE Access*, vol. 5, pp. 26521–26544, 2017.

[2] M. Ayaz, M. Ammad-Uddin, Z. Sharif, A. Mansour, and E. M. Aggoune, "Internet-of-Things (IoT)-based smart agriculture: Toward making the fields talk," *IEEE Access*, vol. 7, pp. 129551–129583, 2019.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[4] T. T. T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 4th Quart., 2008.

[5] R. Passerone, D. Cancila, M. Albano, S. Mouelhi, S. Plosz, E. Jantunen, A. Ryabokon, E. Laarouchi, C. Hegedus, and P. Varga, "A methodology for the design of safety-compliant and secure communication of autonomous vehicles," *IEEE Access*, vol. 7, pp. 125022–125037, 2019.

[6] M. E. Raynor and P. Wilson, *Beyond the Dumb Pipe: The IoT and the New Role for Network Service Providers–The Internet of Things in Telecom.* New York, NY, USA: Deloitte Univ. Press, Sep. 2015. [Online]. Available: https://dupress.deloitte.com/dup-us-en/focus/internet-of-things/iot-in-telecom-industry.html

[7] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Netw.*, vol. 15, no. 2, pp. 24–32, Mar. 2001.

[8] C. Xu, S. Chen, J. Su, S. M. Yiu, and L. C. K. Hui, "A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2991–3029, 4th Quart., 2016.

[9] S. Valenti, D. Rossi, A. Dainotti, A. Pescapè, A. Finamore, and M. Mellia, "Reviewing traffic classification," in *Data Traffic Monitoring and Analysis*, E. Biersack, C. Callegari, M. Matijasevic, Eds. Berlin, Germany: Springer-Verlag, 2013, pp. 123–147. [Online]. Available: http://dl.acm.org/citation.cfm?id=2555672.2555680

[10] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "KISS: Stochastic packet inspection classifier for UDP traffic," *IEEE/ACM Trans. Netw.*, vol. 18, no. 5, pp. 1505–1515, Oct. 2010.

[11] P. Bermolen, M. Mellia, M. Meo, D. Rossi, and S. Valenti, "Abacus: Accurate behavioral classification of P2P-TV traffic," *Comput. Netw.*, vol. 55, no. 6, pp. 1394–1411, Apr. 2011.

[12] G. Cirillo, R. Passerone, A. Posenato, and L. Rizzon, "Statistical flow classification for the IoT," in *Applications in Electronics Pervading Industry, Environment and Society—ApplePies* (Lecture Notes in Electrical Engineering), vol. 627, S. Saponara and A. De Gloria, Eds. Cham, Switzerland: Springer, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-37277-4_9

[13] C. Liu, Z. Cao, Z. Li, and G. Xiong, "LaFFT: Length-aware FFT based fingerprinting for encrypted network traffic classification," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 1–6.

[14] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy, "DEFT: A distributed IoT fingerprinting technique," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 940–952, Feb. 2019.

[15] 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)*, Dec. 2018, pp. 5187–5192.

[16] A. J. Pinheiro, J. de M. Bezerra, C. A. P. Burgardt, and D. R. Campelo, "Identifying IoT devices and events based on packet length from encrypted traffic," *Comput. Commun.*, vol. 144, pp. 8–17, Aug. 2019.

[17] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. San Mateo, CA, USA: Morgan Kaufmann, 2016.

[18] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1988–2014, 2nd Quart., 2019.

[19] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, "A first look at cellular machine-to-machine traffic: Large scale measurement and characterization," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 65–76, Jun. 2012, doi: 10.1145/2318857.2254767.

[20] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," Dept. Comput. Sci., Queen Mary Univ. London, London, U.K., Tech. Rep. RR-05-13, 2005.

[21] A. Bar, P. Svoboda, and P. Casas, "MTRAC–discovering M2M devices in cellular networks from coarse-grained measurements," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 667–672.

[22] M. Laner, P. Svoboda, and M. Rupp, "Detecting M2M traffic in mobile cellular networks," in *Proc. IWSSIP*, May 2014, pp. 159–162.

[23] V. Pant, R. Passerone, M. Welponer, L. Rizzon, and R. Lavagnolo, "Efficient neural computation on network processors for IoT protocol classification," in *Proc. New Gener. CAS (NGCAS)*, Genova, Italy, Sep. 2017, pp. 9–12.

[24] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying IoT traffic in smart cities and campuses," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Atlanta, GA, USA, May 2017, pp. 559–564.

[25] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1745–1759, Aug. 2019.

[26] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis," in *Proc. Symp. Appl. Comput. (SAC)*, New York, NY, USA: Association Computing Machinery, 2017, pp. 506–509.

[27] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for Internet of Things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[28] Y. Yang, K. Zheng, C. Wu, and Y. Yang, "Improving the classification effectiveness of intrusion detection by using improved conditional variational AutoEncoder and deep neural network," *Sensors*, vol. 19, no. 11, p. 2528, Jun. 2019.

[29] A. S. Iliyasu and H. Deng, "Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks," *IEEE Access*, vol. 8, pp. 118–126, 2020.

[30] Y. Zeng, H. Gu, W. Wei, and Y. Guo, "Deep-full-range: A deep learning based network encrypted traffic classification and intrusion detection framework," *IEEE Access*, vol. 7, pp. 45182–45190, 2019.

[31] P. Wang, X. Chen, F. Ye, and Z. Sun, "A survey of techniques for mobile service encrypted traffic classification using deep learning," *IEEE Access*, vol. 7, pp. 54024–54033, 2019.

[32] A. Amouri, V. T. Alaparthy, and S. D. Morgera, "A machine learning based intrusion detection system for mobile Internet of Things," *Sensors*, vol. 20, no. 2, p. 461, Jan. 2020.

[33] X. Han, L. Han, Y. Zhou, L. Huang, M. Qian, J. Hu, and J. Shi, "FFT traffic classification-based dynamic selected IP traffic offload mechanism for LTE HeNB networks," *Mobile Netw. Appl.*, vol. 18, no. 4, pp. 477–487, Aug. 2013.

[34] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "BotFinder: Finding bots in network traffic without deep packet inspection," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, Nice, France, 2012, pp. 349–360.

[35] M. Zhou and S.-D. Lang, "Mining frequency content of network traffic for intrusion detection," in *Proc. Int. Conf. Commun., Netw., Inf. Secur.*, New York, NY, USA, Dec. 2003, pp. 101–107.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[37] H. Guo and J. Heidemann, "IP-based IoT device detection," in *Proc. Workshop IoT Secur. Privacy (IoT S&P)*. Budapest, Hungary: ACM, Aug. 2018, pp. 36–42.

[38] *Mimic MQTT Simulator*. Accessed: Jul. 28, 2020. [Online]. Available: https://www.gambitcomm.com/site/

[39] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," in *Proc. 2nd Int. Conf. Inf. Syst. Secur. Privacy*, Rome, Italy, 2016, pp. 407–414.

[40] A. Dainotti, A. Pescapé, P. S. Rossi, F. Palmieri, and G. Ventre, "Internet traffic modeling by means of hidden Markov models," *Comput. Netw.*, vol. 52, no. 14, pp. 2645–2662, Oct. 2008.

[41] A. Dainotti, A. Pescapé, and G. Ventre, "A cascade architecture for DoS attacks detection based on the wavelet transform," *J. Comput. Secur.*, vol. 17, no. 6, pp. 945–968, Nov. 2009.

[42] AppNeta. (2015). *Tcpreplay: Pcap Editing and Replaying Utilities*. [Online]. Available: http://tcpreplay.appneta.com/

[43] P. Jurkiewicz, G. Rzym, and P. Boryło, "Flow length and size distributions in campus Internet traffic," 2018, *arXiv:1809.03486*. [Online]. Available: http://arxiv.org/abs/1809.03486
[44] L. Grimaudo, M. Mellia, and E. Baralis, "Hierarchical learning for fine grained Internet traffic classification," in *Proc. 8th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Aug. 2012, pp. 463–468.
[45] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" in *Machine Learning and Data Mining in Pattern Recognition*, P. Perner, Ed. Berlin, Germany: Springer, 2012, pp. 154–168.
[46] D. Ho, E. Liang, I. Stoica, P. Abbeel, and X. Chen, "Population based augmentation: Efficient learning of augmentation policy schedules," in *Proc. 36th Int. Conf. Mach. Learn.*, Long Beach, CA, USA, Jun. 2019, pp. 2731–2741.
[47] L. Grimaudo, M. Mellia, E. Baralis, and R. Keralapura, "SeLeCT: Self-learning classifier for Internet traffic," *IEEE Trans. Netw. Service Manag.*, vol. 11, no. 2, pp. 144–157, Jun. 2014.
[48] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA?" in *Proc. IEEE 20th Int. Symp. Field-Program. Custom Comput. Mach.*, Apr. 2012, pp. 232–239.
[49] D. R. Butenhof, *Programming With POSIX Threads*. Reading, MA, USA: Addison-Wesley, 1997.
[50] Mellanox. (2017). *Mellanox Indigo NPS-400 Network Processor*. [Online]. Available: https://www.mellanox.com/related-docs/prod_npu/PB_Indigo_NPS-400.pdf

GENNARO CIRILLO received the B.S. and M.S. degrees in computer science from the University of Trento, Italy, in 2017 and 2020, respectively. His research interests include formal methods, logic and optimization, automata theory, machine learning, and the IoT technologies.

ROBERTO PASSERONE (Member, IEEE) received the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley, in 1997 and 2004, respectively. He was a Research Scientist with Cadence Design Systems. He is currently an Associate Professor of electronics with the Department of Information Engineering and Computer Science, University of Trento, Italy. He has published numerous research articles in international conferences and journals in the area of design methods for systems and integrated circuits, formal models, and design methodologies for embedded systems, with particular attention to image processing and wireless sensor networks. He has participated in several European projects on design methodologies, including SPEEDS, SPRINT, and DANSE. He was the Local Coordinator for ArtistDesign, COMBEST, and CyPhERS. He has served as the Track Chair of the real-time and networked embedded systems with ETFA, from 2008 to 2010, and the General Chair and the Program Chair for various editions of SIES.