A Classification-Based Algorithm to Detect Forged Embedded Machines in IoT Environments

Valerio Selis, *Member, IEEE*, and Alan Marshall, *Senior Member, IEEE*

***Abstract*—In the Internet of Things (IoT), interconnected devices manage essential information related to people's lives; hence, securing this information is essential. The number of these machines is rapidly growing; these are mostly embedded, and therefore more susceptible to attacks. Recently, thousands of subverted IoT embedded machines, such as surveillance cameras, were used for launching distributed denial of service (DDoS) attacks. In this scenario, attackers, who are not embedded machines, can emulate their behaviors to subvert the machine-to-machine network. In this paper, we present a novel algorithm to detect such forged machines. This allows detection of virtualized and emulated systems by observing their behaviors and can be used by IoT trust agents in embedded machines. With the aim of creating a machine-agnostic system, portable and applicable to future IoT machines, we propose a classification-based algorithm as the detection mechanism. Extensive experiments with different system architectures and operating systems were performed, along with a comparison of several feature selection and classification methods. The results show that our method can quickly reveal illegitimate machines with a high probability of detection, giving the opportunity for its use in power-constrained machines. Our approach is also able to detect unknown embedded systems and can be used to detect fake timing attacks.**

***Index Terms*—Classification methods, Internet of Things (IoT), machine-to-machine (M2M) communications, timing analysis, trust, virtual machining.**

I. Introduction

NOWADAYS, there are between 5 and 15 billion devices already connected to the Internet, and it is expected there will be 21 to 50 billion by 2020, of which 13 billion are likely to be wirelessly connected [1]–[3]. These "things" have different capabilities and complexities, starting from a simple smart sensor to a high-end supercomputer. In the Internet of Things (IoT) paradigm, their collaboration is important in order to reach specific and complex goals. The devices communicate to each other, and some are connected to the Internet, including but not limited to, virtual and real objects. In this study, the term "object" means everything that is not directly related to human beings. At present, there are various applications that have started to emerge across several fields, such as healthcare, smart robots, intelligent transportation systems, manufacturing systems, smart building technologies, and smart grids [4]–[8]. These IoT application scenarios are widespread and so it is not practical to implement conventional "enterprise"-type security measures, as for example, many IoT devices will frequently encounter new devices that are not part of a corporate system. This factor is enhanced whenever public networks are used for data offloading and IoT devices share information independently of the communication infrastructure, location, and software applications used [9], [10].

In this context, real objects have the important role of managing the data collected from real environments. Most importantly, the information gathered from the environment will be used to actively give feedback and very likely, to intervene in people's lives in an autonomous way. For these reasons, their security is a primary aspect that needs to be addressed [11]. Moreover, it is widely recognized that most IoT devices are based on embedded systems [4], [5]. Embedded machines generally have low computational capability and are more susceptible to attacks. In fact, to detect sophisticated attacks, machines need to analyze a large amount of information in a short period of time. However, a machine with low computational capability is not able to process this amount of information quickly enough to detect an attack in real time or to detect it at all. An embedded machine is therefore more susceptible to attacks compared to a machine with high computational capabilities. In other words, this means that embedded machines can be easily compromised, and as such, can be a target for attackers.

From the real objects' perspective, there are two main types of communications:

1) machine-to-machine (M2M) communication;
2) machine-to-human (M2H) communication.

These types of communications can be subverted by an attacker in order to create the following types of communications:

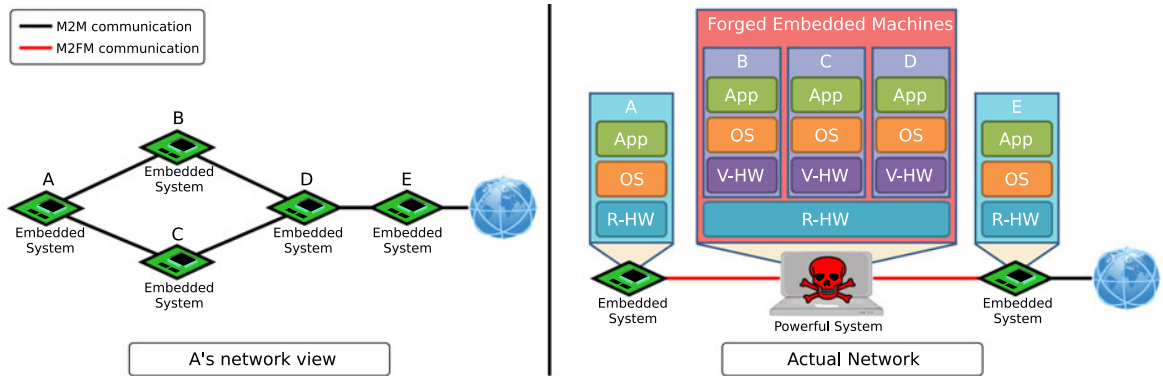
1) machine-to-fake machine (M2FM) communication, and vice versa;
2) machine-to-fake human (M2FH) communication, and vice versa, in which a fake human is a machine mimicking human behaviors, e.g., a robot with an AI capability of reproducing human behaviors.

Trust has an important role in securing against these subverted communications. In fact, several trust management frameworks (TMFs) have been proposed [12]–[18]. Nitti *et al.* [15]–[18] incorporate different trust weight values in their TMFs, for devices with different computational capabilities as shown in Table I. As shown in this table, these devices are mainly embedded, and by considering M2FM communications, an attacker may prefer to forge devices such as smartphones, tablets, and/or laptops. This

Manuscript received March 26, 2017; revised November 3, 2017; accepted March 30, 2018. This work was supported in part by the Engineering and Physical Sciences Research Council under Grant 1566989. *(Corresponding author: Valerio Selis.)*

The authors are with the Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool L69 3GJ, U.K. (e-mail: Valerio.Selis@liverpool.ac.uk; Alan.Marshall@liverpool.ac.uk).

Digital Object Identifier 10.1109/JSYST.2018.2827700



(a) (b)

Fig. 1. Threat model with a representation of multiple forged embedded machine attack in IoT in order to subvert M2M communications. (a) This shows A's view of the network from which there are apparently no issues. (b) This shows the actual network topology including the attacker faking B, C, and D.

TABLE I
ASSIGNED TRUST WEIGHT VALUES DEPENDING ON THE COMPUTATIONAL CAPABILITIES OF MACHINES

Device	Trust weight value	
RFID	0.2 [15], [17]	Node resources in percentage (0 to 100%) [16]
Sensor	0.2 [17], [18], 0.4 [15]	
Recorder	0.2 [18]	
Set-top box	0.4 [18], 0.6 [15], 0.8 [17]	
Smart video camera	0.4 [18], 0.6 [15]	
Smart gateway and terminal	0.6 [18]	
Smartphone and tablet	0.8 [15], [17], [18]	
Laptop	0.8 [18]	

is because these devices have a high trust weight value assigned during the trust computation by the TMF and, therefore, their initial trust value will be higher.

In M2FM communication scenario, a real machine aims to access to the Internet by communicating with other machines. Therefore, it will start to trust them accordingly, assuming its neighbors are real. However, this machine is not able to assess if the other machines are real or not. An attacker can use multiple virtual or emulated systems in order to create forged embedded machines and consequentially compromise the network or part of it, as shown in Fig. 1. Fig. 1(a) shows an example in which a real embedded machine "A" assumes that it is communicating with other real embedded machines "B," "C," "D," and "E" and will trust them accordingly. However, as shown in Fig. 1(b), "A" is communicating with an attacker faking "B," "C," and "D" and creating an M2FM communication. This could allow the attacker to collect data transmitted by "A" and to give to "A" false recommendations about "E." This may lead to security issues, in particular for the data transmitted throughout the attacker. An attacker can create a large amount of forged embedded machines in order to attack the TMF. In this case, multiple forged embedded machines in the network can attack it simultaneously by launching several attacks such as good-mouthing attacks, self-promoting attacks, on–off attacks, etc. For example, to launch a good-mouthing attack, node "B" can increase the reputation of node "C" by providing good recommendations to "A." To launch the attack using multiple forged embedded machines, a powerful machine (with greater computational resources) may be more preferable than an embedded machine. It can also misinform "A" and/or poison the information it sends or receives, e.g., Man-in-The-Middle attacks.

The aim of this paper is to remove the opportunity for an attacker to create M2FM communications by detecting forged embedded machines in the network. The method used to detect these illegitimate machines can be used by trust agents to secure the M2M communication. This will give the opportunity to create trust relationships among devices in the network.

The main contributions of the paper are as follows:

1) Analysis and adoption of machine learning algorithms in order to easily detect forged embedded machines in the network.
2) Definition of a novel method to detect unreal embedded machines quickly with a high detection rate, which is independent of the network topology and architecture.
3) A large number of real, virtual, and emulated embedded systems have been systematically tested in order to produce a valuable dataset to be used as a reference for the detection.
4) Demonstration of the advantage of this solution and its easy applicability to future IoT devices, by creating trust relationships in M2M communications and removing M2FM communications from the network.
5) Testing the reliability of the detection method when applied to unknown embedded systems (UESs).

The rest of this paper is organized as follows. In Section II, the work related to the detection of virtual and emulated systems is presented. In Section III, we describe the proposed solution by comparing different feature selection and classification methods. The results of these comparisons are shown and discussed in Section IV. The proposed solution is tested against UESs in Section V. Finally, in Section VI, we present our conclusion and future research directions to apply this method to TMFs in IoT.

II. RELATED WORK

Research work on detecting forged embedded machines in M2M communications and IoT is at an early stage and, as far as

we are aware, the work proposed by Selis and Marshall in [19] is the first addressing this specific problem. Related works have focused on the detection of specific virtual and emulated systems, in some cases, by using specific architecture-dependent information from $\times 86/\times 64$ architectures. The detection of embedded emulated systems like the Dalvik VM for Android-based systems is considered in [20] and [21] by using fingerprinting tests. In the following sections, the detection mechanisms adopted in the literature to detect virtual and emulated systems are shown.

A. CPU and Memory Tests

Information from the memory and CPU registers is gathered for the detection [22]–[24]. These tests detect unreal machines only if the architecture is based on the Intel Architecture at 32 bit. Moreover, if the virtual or emulated systems use the real CPU, these tests can fail. For these reasons, CPU and memory tests cannot be adopted to detect forged embedded machines in IoT, because there are several different architectures, such as embedded $\times 86$, MIPS, ARM, PowerPC, etc. It is not feasible to test all of these architectures to find, if any, specific information from CPU and memory that can lead to the detection of virtual and emulated environments. Furthermore, if there is a new architecture, it needs to be tested in order to verify if there is a possible way to use CPU and memory registers for the detection. One way forward is to use system-on-chip with customized architectures. However, this approach means that there will be no previous knowledge of the architecture, and so these detection methods are not feasible.

B. Architecture-Based Timing Tests

Specific CPU instructions are used to carry out a time analysis and the information obtained is used for the detection [25], [26]. These tests are based on measuring the time to access specific CPU registers present in $\times 86/\times 64$ architectures by using read and write instructions. As stated, these are architecture-dependent operations, and difficult to apply across all the processor architectures encountered in the IoT environment. In fact, the detection should be done by treating the machine architecture as a black box.

C. Remote Tests

Information from the TCP packets in the network is used for the detection [27], [28]. Remote tests are based on using the TCP timestamp option in TCP packets. This option is used by default only in Linux-based machines and it is used to calculate the clock skew of a system. Polčá *et al.* in [29] and [30] have demonstrated that it is possible to easily remove it by using the network time protocol (NTP) daemon. It is also possible to fake it by mimicking a clock skew of another device as demonstrated in [31].

D. Fingerprinting Tests

Specific "signature" from the system are collected for the detection [20], [21], [25], [26], [28], [32]. Fingerprinting mechanisms collect information from the system such as driver names, running applications, system registry keys, hardware IDs such as MAC-addresses, system APIs access where available, etc. This information can be easily faked in virtual and emulated environments, especially with open source OS. For example, we were able to modify the kernel in order to display fake CPU information that can be obtained from/proc/cpuinfo. This can be also done for modules, drivers, and applications by changing their information and recompiling them. It is also very easy to fake the MAC address of network interfaces by adopting the MAC-Changer application [33].

E. Behavioral Tests

System behaviors may also be used for detection. A behavior test has been demonstrated to be efficient for detecting virtual and emulated systems. In the work presented in [19], a machine emulation detection algorithm is presented. This uses behaviors from real, virtual, and emulated embedded systems for the detection. However, its main limitation is that the machine is represented with a feature space of one feature per time by applying predefined threshold values. This can lead to a misclassification of real embedded machines, or for forged embedded machines to go undetected. Moreover, the time required to detect virtual or emulated embedded systems in a network is very high (around 3.5 min). These performances mean that these tests are very unlikely to be applied in IoT environments for three main reasons: 1) high detection uncertainty; 2) slowness in the detection; and 3) open to fake timing attacks, such attacks consist of using powerful machines with a modified kernel in order to fake the timing behaviors. These are very important factors for detecting in mobile environments, and for devices with power-constrained resources.

III. Proposed Solution

In the previous section, we underlined that current detection methods cannot be used for detecting forged embedded machines in the IoT environment. For this reason, a new method is proposed. Virtual and emulated embedded systems (VESs) are normally used by developers and researchers as test environments. These are almost exclusively used to test and run applications without having a real embedded hardware. In this case, a VES needs to translate all instructions from the host architecture to the virtualized or emulated system, called guest. These translations are needed for each part of the embedded system, such as CPU registers and instructions, memory management, physical devices management, etc. The execution of a set of instructions in a VES should be the same as the execution in a real embedded machine (REM). However, VESs introduce various behaviors that cause differences in system resources availability, timing dependencies, and I/O devices communication [34], [35]. In this paper, timing differences between REMs and VESs are used as behaviors characterization method.

A. Detection Model

The detection model is based on "IoT trust agents" running on IoT devices and capable of communicating with each other

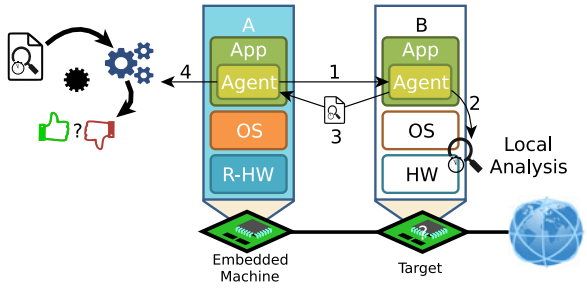


Fig. 2. Representation of the detection model. Agent "A" requests to agent "B" to run the characterization algorithm locally and starts counting the elapsed time (1). Agent "B" performs the local characterization (2) and then it sends back the timing results to agent "A" (3). Agent "A" estimates the round-trip times, stops the counter, and performs the final detection (4).

inside the IoT architecture. From now on in this paper, these will be called "agents." Each agent gives the opportunity to another agent in the network to run the characterization algorithm locally as shown in Fig. 2.

The characterization algorithm consists of pinging localhost (127.0.0.1) in device "B" n times with an interval between pings of 0.2 s, this interval is used to speed up the characterization time. The number of pings required to obtain a good detection will be determined by changing the n value. Also, as expected, reducing the number of pings leads to a corresponding decrease in the overall detection time. For each ping, timing information, such as ping response time (P) and system timestamp (T), is collected. T is used as control time information. If an attacker tries to fake P, the algorithm will rely on the changes in T and vice versa. In fact, an attacker that tries to fake P and/or T values needs to implement additional functions to fake the ping. These new functions need to be translated from the host architecture to the virtualized or emulated system and then executed. These result in an increase in the time between two pings (T) and the characterization time, leading to a detection.

There are several reasons why the ping command is adopted. First, it is available in all systems that support connectivity. Second, it provides precise timing information related to the networking stack. In fact, when a VES uses the ping command, it needs to create ping request and response packets, and manage these packets in the kernel and user space. All these operations need to be done by translating every single instruction from the host architecture to the guest architecture. The rationale is that all these tasks are going to cause a change in the timing behaviors. A third reason for using the ping command is that it makes it difficult for a VES to fake the behavioral results, because this command uses the network stack in order to perform the characterization. If the method presented in here is applied, VESs would need to check when a socket is created every time. This task allows checking that the socket is created by the characterization algorithm. This leads to an increase in the overall delay time and decrease in system performances. Additionally, this requires a change to part of the kernel of a VES which is not always possible. Finally, if the VES tries to slow down its time in order to fake the characterization, the agent that requests the characterization can use its local time to detect this. To do so, it has to simply measure the difference between the request (T_{Req}) and the response (T_{Resp}) of the characterization as follows:

$$T_{\mathrm{Resp}} - T_{\mathrm{Req}} \approx T_{\mathrm{Ch}} + \mathrm{e}(\mathrm{RTT}_{\Delta T_{\mathrm{Ch}}}) \tag{1}$$

where T_{Ch} is the characterization time

$$T_{\mathrm{Ch}} = n_{\mathrm{pings}} \cdot 0.2 \text{ s} \tag{2}$$

and $\mathrm{e}(\mathrm{RTT}_{\Delta T_{\mathrm{Ch}}})$ is the estimated round-trip times after T_{Ch} seconds are elapsed. For example, if the characterization is supposed to take 1 min, the difference should be around 1 min plus network communication delays and not less than that.

Therefore, we apply several supervised learning methods in which each machine is identified with two or more features per time. The following sections highlight the steps adopted to efficiently detect forged embedded machines in the network.

TABLE II
LIST OF REAL, VIRTUAL, AND EMULATED EMBEDDED SYSTEMS

System	Architecture	Operating System
ALIX 6F2 [a]	Embedded x86	Debian
Android Emulator [c]	ARMv7	Android
Arduino Yún [a]	MIPS	OpenWRT
Carambola [a]	MIPS	OpenWRT
Genymotion [c]	Embedded x86	Android
Google Nexus 5[a]	ARMv7	Android
Google Nexus 7[a]	ARMv7	Android
GXemul [c]	MIPS	NetBSD
Netgear Centria WNDR4700 [a]	PowerPC	OpenWRT
Open Mesh OM2P-LC [a]	MIPS	OpenWRT
OVPsim [c]	MIPS	Debian
QEMU [c]	ARM6l	Debian
QEMU [c]	MIPS	OpenWRT
QEMU [c]	MIPSel	OpenWRT
QEMU [c]	PowerPC	OpenWRT
Raspberry Pi [a]	ARMv6l	Debian
Samsung Chromebook Series 3[a]	ARMv7	Chrome OS
Samsung Galaxy Tab GT-P1000 [a]	ARMv7l	Android
VirtualBox [b]	Embedded x86	OpenWRT
VMware [b]	Embedded x86	OpenWRT

[a]Real embedded devices; [b]Virtual embedded system; [c]Emulated embedded system.

B. Dataset

The dataset is composed of information gathered from real, virtual, and emulated embedded systems by using the characterization algorithm. Table II shows the systems considered here. In order to create the dataset, 1000 tests for each system were performed, half with normal CPU usage and half by stressing the CPU (CPU usage around 100%). By varying the CPU usage, it is possible to understand if the detection of forged embedded machines is affected by unpredictable changes in system performances. In fact, a machine could be able to run several applications at the same time that may affect its behaviors. However, the CPU usage cannot be used to detect VESs, because it can be modified in real time by running different applications. In summary, a total of 20 000 tests were carried out, out of which 10 000 were obtained from REMs and 10 000 from VESs. For each test, information about the ping response time (P) and timestamp (T) was collected and used as characterization metrics.

TABLE III
CHARACTERIZATION FEATURES

Feature	Equation
Minimum	$\min = min(X)$
Maximum	$\max = max(X)$
Range	$\text{range} = \max - \min$
Sum	$sum = \sum_{i=1}^{N} x_i$
Mean	$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$
Variance	$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$
Standard deviation	$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$
Upper/Lower bounds (95%)	$U/L = \mu \pm 1.96\,\sigma$
Skewness	$SKEW = \frac{\sum_{i=1}^{N}(x_i - \mu)^3}{N\sigma^3}$
Kurtosis	$KURT = \frac{\sum_{i=1}^{N}(x_i - \mu)^4}{N\sigma^4}$
Mode	$mode = freq(X)$
Median	$median = \frac{1}{2}\left(x_{\frac{N}{2}} + x_{\frac{N}{2}+1}\right) \text{of } sort(X)$
Pearson correlation coefficient	$r = \frac{\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{N}(x_i - \mu_x)^2}\sqrt{\sum_{i=1}^{N}(y_i - \mu_y)^2}}$

X = all samples; x = ith sample; N = number of samples; *freq*= most frequent value; *sort* = values in sorted order.

C. Features Extraction

The information present in the dataset is related to P and T. In order to calculate the variability of these measurements, statistical methods were adopted, as shown in Table III. These characterization features are used to understand how the timing behaviors change over time. In particular, features obtained from the central tendency, dispersion, and distribution of the data are used. This choice is based on the hypothesis that, considering the longer time VESs can take to perform the translation of all instructions, these features are likely to give more accurate results in detecting them. A total of 28 features were extracted, 14 for P and 14 for T.

At this stage, all features are preprocessed by adopting the Min–Max normalization method calculated as follows:

$$\tilde{x}_i^j = \frac{x_i^j - \min^j}{\max^j - \min^j} \tag{3}$$

where x_i^j is the ith value of the jth feature and, $\max^j$ and $\min^j$ are respectively the maximum and minimum values of the jth feature. By using this method, all features are normalized within values between 0 and 1 in order to avoid zero entries in sparse data and increase the strength of features selection and classification methods against small standard deviations of features.

D. Features Selection

The feature selection step is used to select the most important features and decrease the computational time and complexity to reach the detection. Our hypothesis is that REMs will have similar timing behaviors during their functional operation. The results from P and T are therefore expected to follow a specific data distribution and we assume that central tendency features will have an important role in detecting forged embedded machines. Different features selection methods were adopted in order to obtain the best result and these are as follows:

1) *Select-K-Best:* select features according to the k highest scores with both ANOVA F-value (SKB-F) and Chi-squared stats of nonnegative features (SKB-Chi2) [36].
2) *Extremely randomized trees* (ERT) [36], [37].
3) *Recursive feature elimination* (RFE) [36], [38].
4) *L1-based feature selection* (L1-FS) [36], [39].

These methods were applied in order to extract the best kth features from P and the best kth features from T, where k is a value between 1 and 14. It is very important to select always at least one feature from P and one from T. In fact, if there are only the best kth features of a specific measure and an attacker fakes it, this may lead to a misdetection.

E. Classification

The classification step is a process used to identify if an observation belongs to a specific category. In this case, it is used to detect if an unknown embedded machine (observation) is real or unreal (categories). This step is subdivided into two main steps: learning and testing. In the first step, the classifier is trained with a dataset, called training set, in which the class is known for each sample. In this step, the classifier learns how to recognize which sample belongs to a specific class. Moreover, a cross-validation over the training set is applied in order to select the best parameters for the classification methods, a tenfold cross-validation is used. In the second step, the classifier uses its experience to classify new data, called test set. Its classes for each sample are known but not used by the classifier. In our experiment, the dataset was subdivided into 75% for the training set and 25% for the test set.

The following supervised learning methods for classification problems were adopted:

1) *Decision Tree* (DT) [36], [40]–[42].
2) *Naive Bayes* (NB) [36], [43].
3) *Stagewise Additive Modeling using a Multi-class Exponential loss function* (SAMME) [36], [44].
4) *Random Forest* (RF) [36], [45].
5) *Support Vector Machine* (SVM) [36], [46], [47]. Three different kernels were adopted: linear (L-SVM), polynomial (P-SVM), and radial basis function (R-SVM). These algorithms use two parameters in order to obtain the maximum accuracy such as C and gamma. The first one is the penalty for misclassification and the second one is the deviation of the kernel. C was sampled at 10^{-2}, 10^{-1}, ..., 10^{3}, while gamma at 10^{-9}, 10^{-8}, ..., 10^{3}.
6) *K-Nearest-Neighbor* (k-NN) [36]. The number of k neighbors was sampled at 1, 2, ..., 30.
7) *Linear Discriminant Analysis* [36], [44].
8) *Quadratic Discriminant Analysis* [36], [44].

F. Performance Evaluation

The performance evaluation is an important tool to check how well the classification methods classify the test set. This is used to understand the performance of classifiers in detecting forged embedded machines in the network.

For the evaluation, four different measures were used for each classifier:

1) *Accuracy* (A)*:* measures how well the classifier is able to classify the samples.
2) *Precision* (P)*:* measures the percentage of samples that are classified as positive and are really positive.
3) *Recall* (R)*:* measures the percentage of positive samples that are classified as positive.
4) *F1-score* (F1)*:* this is the harmonic mean considering both precision and recall.

These measurements return values between 0 and 1, where 1 is the best result. Measurements such as precision, recall, and F1-score are calculated for both real and forged embedded machines. Therefore, the average of these measurements from REM and VES is calculated to have a final value to define the performance of each classifier, termed overall detection performance (ODP) as

$$\mathrm{ODP}_i^j = \frac{A_i^j + \sum_{m=0}^{1} P_{m_i}^j + R_{m_i}^j + F1_{m_i}^j}{7} \tag{4}$$

where m is 0 (REM) or 1 (VES), i is the ith combination of features selection method and classifier, j is the jth tuple of features selected (kP, kT), where k is the number of the kth best features, i.e., with k equal to 3, (3P, 3T) is a tuple with the third best features from ping response time and the third best features from timestamp. In this study, we do not consider the combination of tuples of features like (kP, lT), where k could be different from l, i.e., (3P, 2T) tuple.

The final evaluation is obtained by considering the characterization time (T_{Ch}), the features extraction time required for extracting a tuple of best features from a target (T_{FE}), the classification time (T_{Cl}) and ODP. The features selection time is not considered for the final evaluation as it will not be present during the detection. For this reason, we define the overall detection speed (ODS) for a possible target as

$$\mathrm{ODS}_i^j = T_{\mathrm{Ch}} + \frac{T_{\mathrm{FE}}^j + T_{\mathrm{Cl}_i}^j}{N} \tag{5}$$

where N is the number of samples in the test set, i is the ith combination of features selection method and classifier, and j is the jth tuple of features selected (kP, kT), where k is the number of the kth best features.

G. Overall Evaluation

In order to find the best solution for every different number of pings, the minimum ODS and the maximum ODP are calculated for each tuple of features and each combination of features selection methods and classifiers as

$$\mathrm{ODPmax}_i^j = \max(\mathrm{ODP}_i^j) \tag{6}$$

$$\mathrm{ODSmin}_i^j = \min(\mathrm{ODS}_i^j). \tag{7}$$

TABLE IV
EVALUATION SCORES FOR ODP AND SPEED

Ranges	$\mathrm{ODSscore}_i^j$
$ODS_i^j = ODSmin_i^j$	5
$ODSmin_i^j < ODS_i^j \leq ODSmin_i^j + 0.001s$	4
$ODSmin_i^j + 0.001s < ODS_i^j \leq ODSmin_i^j + 0.01s$	3
$ODSmin_i^j + 0.01s < ODS_i^j \leq ODSmin_i^j + 0.1s$	2
$ODSmin_i^j + 0.1s < ODS_i^j \leq ODSmin_i^j + 1s$	1
$ODS_i^j > ODSmin_i^j + 1s$	0
Ranges	$\mathbf{ODPscore}_i^j$
$ODP_i^j = ODPmax_i^j$	5
$ODPmax_i^j < ODP_i^j \leq ODPmax_i^j - 0.01\%$	4
$ODPmax_i^j - 0.01\% < ODP_i^j \leq ODPmax_i^j - 0.1\%$	3
$ODPmax_i^j - 0.1\% < ODP_i^j \leq ODPmax_i^j - 1\%$	2
$ODPmax_i^j - 1\% < ODP_i^j \leq ODPmax_i^j - 10\%$	1
$ODP_i^j > ODPmax_i^j - 10\%$	0

i=ith combination of feature selection methods and classifiers; j=jth tuple of features selected (kP, kT), where k is the number of features.

Then, a score is assigned to ODP and ODS between 5 and 0 for each tuple of features, and each combination of features selection methods and classifiers are shown in Table IV, in which, the value 5 represents the best result.

In this study, we give more importance to ODP than ODS with respective ratios of 70% and 30%. This is because more relevance is given to the proper identification of forged embedded machines in a short time frame. In fact, by increasing the ODS percentage and therefore saving time, it may happen that a lower percentage of ODP is selected and this will increase the uncertainty during the trust evaluation.

The final evaluation score (FES) is then calculated as follows:

$$\mathrm{FES}_i^j = \frac{\mathrm{ODPscore}_i^j \times 70}{\max(\mathrm{ODPscore}_i^j)} + \frac{\mathrm{ODSscore}_i^j \times 30}{\max(\mathrm{ODSscore}_i^j)}. \tag{8}$$

In this case the value of a FES is between 0 and 100, where the maximum value represents the best combination of features selection and classification method for that features selected.

The best features selected are identified by using the rank of a set as follows:

$$\mathrm{rank}\,(X_i) = \bigcup \begin{cases} \mathrm{rank}\,(y_0) = 1 \\ \mathrm{rank}\,(y_j) = \mathrm{rank}\,(y_{j-1}) + 1 \end{cases} ; \text{for } j = 1, \ldots, n \tag{9}$$

where X_i is a sorted set composed of the sum of how many times each feature is selected; by considering the ith combination of features selection method and classifier that gives as result the highest FES, y_j is the jth element in X_i and n is the number of elements in X_i.

IV. SIMULATION AND RESULTS

Simulations have been performed by using Python and the Scikit-learn module [36]. The information gathered from VES was obtained by using virtual and emulated applications with default configurations in a Linux-based system with a quad-core i3 CPU at 3.40 GHz and 8 GB of RAM.

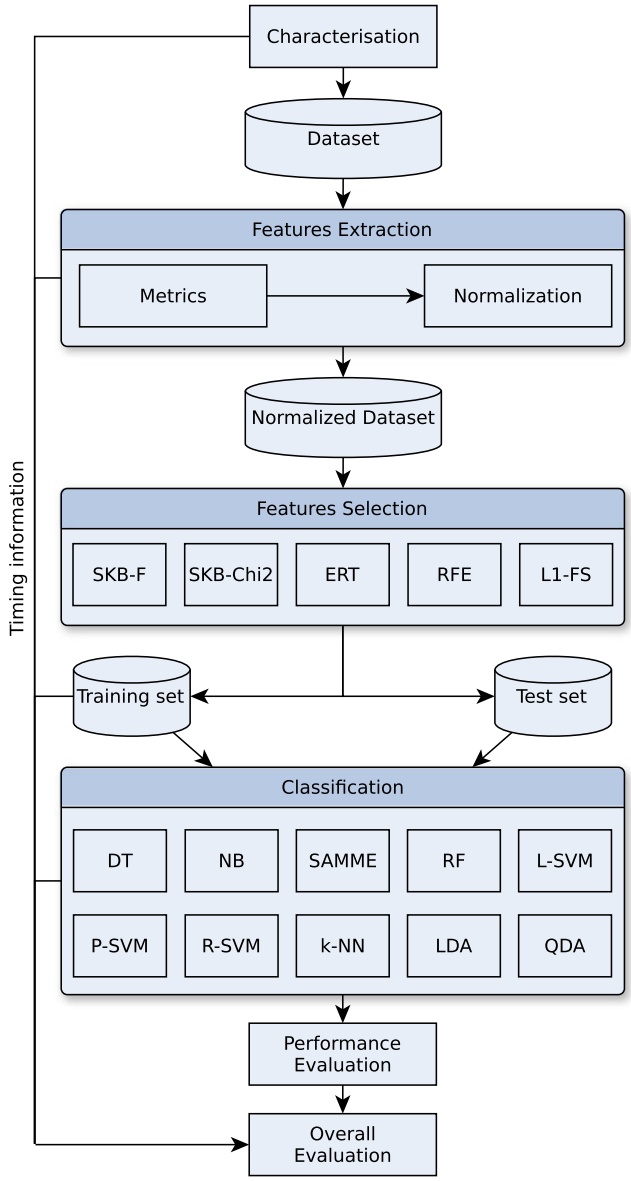


Fig. 3. Steps required for selecting the best combination of features selection and classification method.

In the following part, results obtained from each step used to select the best combination of features selection method and classifier are presented. In the following, we show the results obtained to perform the final detection of forged embedded machines.

Fig. 3 shows a summary of steps required to choose the best combination of features selection and classification methods in order to detect forged embedded machines. These steps were carried out for a total of seven times by changing the number of pings each time, i.e., 1000, 500, 200, 100, 50, 25, and 15.

The timing information related to features extraction and characterization steps is shown in Table V. It shows the decreasing of these times for different number of pings, which is very important in order to speed up the detection.

Fig. 4 shows the maximum time required by each features selection method in order to select a tuple of best features from the training set and for different number of pings. This shows that the quickest features selection method is ERT. The same timing information related to classification methods is shown in Fig. 5, in which the quickest classification method is *k*-NN by considering 1000 pings. Comparing the information from

TABLE V
TIMING INFORMATION RELATED TO THE CHARACTERIZATION AND FEATURES EXTRACTION STEPS

Number of pings	Characterisation time (s)	Extraction of all features from the dataset (s)	Extraction of all features from a target (s)
1000	200	43.229	0.002
750	150	35.715	0.002
500	100	29.558	0.001
250	50	23.304	0.001
100	20	18.778	0.001
50	10	18.100	0.001
25	5	17.651	0.001
15	3	16.651	0.001

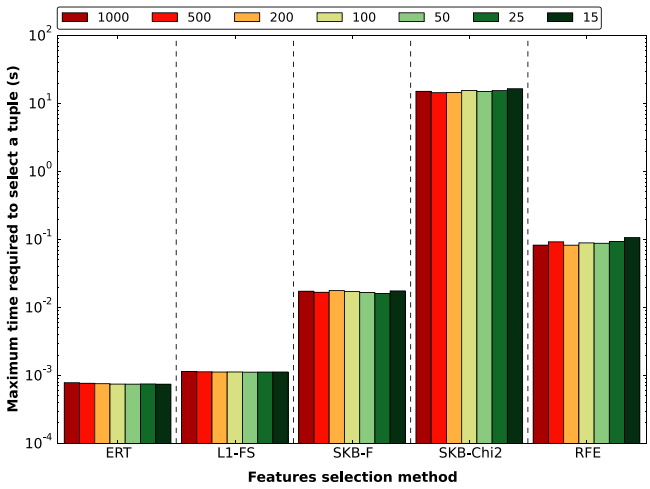


Fig. 4. Maximum time required by features selection methods to select the best features from the training set and for different number of pings.

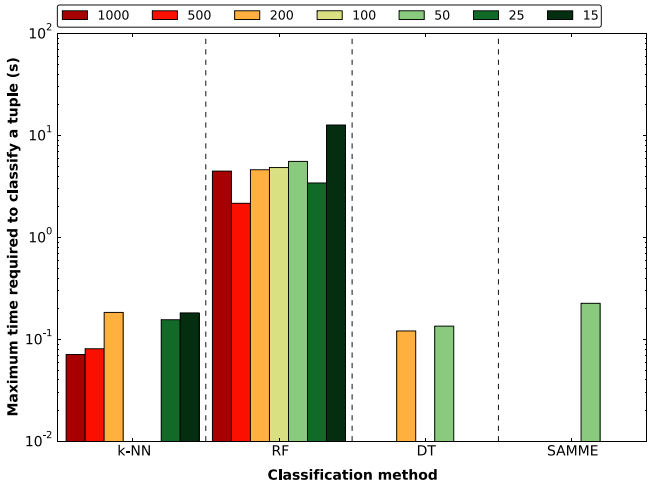


Fig. 5. Maximum time required to classify all data present in the training set for different number of pings.

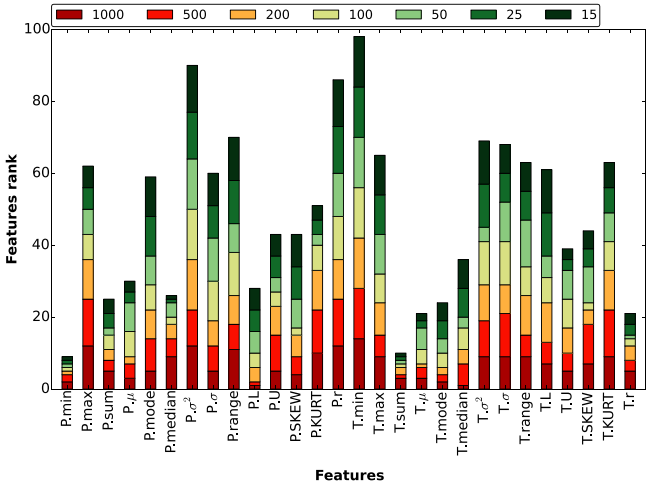


Fig. 6. Rank of how many times the features were selected by the best features selection method for different number of pings. P. refers to ping response time, T. to timestamp, and features abbreviations from Table III are used. For example, P.L means lower bound value of ping response time.

Table V, and Figs. 4 and 5, it is possible to see that the limiting factor is the characterization time.

After the features selection step, the rank of the features selected for each number of pings is calculated as shown in Fig. 6. The lower value represents the best feature chosen. It is possible to see that features such as P.min and T.sum are very useful for detecting forged embedded machines as the sum of their ranks is very low.

Fig. 7 shows which features selection methods were adopted in order to select the best tuple of features for different number of pings. SKB-F works well to select the best two and four features. L1-FS selects mostly the best 10 to 18 features. ERT selects the best 2 to 26 features for different number of pings. SKB-Chi2 works well in selecting the best 22 features. Finally, RFE is adopted only five times (2, 4, 6, 24 and 26 features) for 100 and 200 pings. The (14P, 14T) tuple is not shown in the figure as all features are used, therefore, in this case this tuple is not needed for the selection.

Fig. 8 shows which classification methods return the highest FES value. These methods were adopted in order to classify the best tuple of features for different number of pings. *k*-NN classifies real and forged embedded machines by using two and four features for high and small number of pings. The RF results show it to be the best classification method to properly classify real and forged embedded machines for a wide range of number of pings. DT and SAMME were adopted only few time each. All the other classification methods returned a lower FES value. For this reason, these methods are not shown and are not considered for classifying.

Fig. 9 shows the information about ODP for different numbers of pings and feature tuples. In this figure, it is possible to see that there is a saturation point when the best ten features are selected. The results obtained from the combination of Figs. 7–9 are summarized in Figs. 10 and 11. In these figures, the best ten features which give the best FES from ERT and L1-FS methods are shown in combination with RF and *k*-NN. Information about the classification time is shown in Fig. 11. The classification time is reported instead of ODS because the characterization time is the limiting factor. This figure shows that *k*-NN is faster than RF, however, RF is more reliable than *k*-NN for detecting forged embedded machines as shown in Fig. 10.

As a result, it is possible to obtain an ODP value greater than 99.5% with only 25 pings with L1-FS and RF. The best ten features obtained by using L1-FS are P.min, P.median, P.L, P.SKEW, P.KURT, T.sum, T.μ, T.range, T.L, and T.SKEW. Moreover, by using ERT with RF for 200 pings, the ODP value is 99.9% circa. In this case, the best ten features selected are P.min, P.sum, P.μ, P.median, P.L, T.sum, T.mode, T.median, T.U, and T.r. It is clear that by increasing the number of pings, there is more information available and that different features can be more accurate in detecting a forged embedded machine, while some features only introduce uncertainty. Finally, it is important to note that by adopting these combinations, the value of ODS for 25 pings is equal to approximately 5 s; for 200 pings it is around 40 s. This shows the high efficiency of our method in terms of time required for the detection and the high detection performance.

V. Detection of Unknown Systems

The final detection method is also tested against UESs in order to check its performances on recognizing REMs and VESs that are not present in the dataset. Two virtual embedded systems, VMWare ESXi and VirtualPC, and one real embedded system, the Zsun WiFi Card Reader were used. These systems are very different from other systems present in the dataset. VMWare ESXi is a hypervisor that runs directly on the physical hardware, compared to other virtual or emulated systems that run on a host Linux-based OS. VirtualPC is a virtual machine that runs under a Windows-based OS. Finally, the Zsun WiFi Card Reader is a card reader that provides access to files via WiFi, while other real embedded systems in the dataset were development boards, smartphones, and tablets with more resources and capabilities. The same amount of characterization tests was carried out for all these UESs. Fig. 12 shows the results obtained by using the final detection method for 25 and 200 pings. It is possible to see that, with 25 pings and L1-FS with RF, there is approximately 25% probability of recognizing an unknown REM as a VES, and a very low uncertainty in recognizing unknown VESs. Meanwhile, with 200 pings and ERT with RF, an unknown REM is correctly detected as well unknown VESs. These results show that this method can still detect REMs and VESs that are not present in the dataset with a very high accuracy.

Power consumption, CPU, and memory usage of the characterization algorithm have been evaluated using 200 pings. For this evaluation, the Raspberry Pi was used as a reference system. The VES used to forge the reference system was QEMU. The Linux-based system used during the simulations in Section IV has been employed for running QEMU. Results of this evaluation are summarized in Table VI. It is possible to see that the VES requires more CPU and power for running the

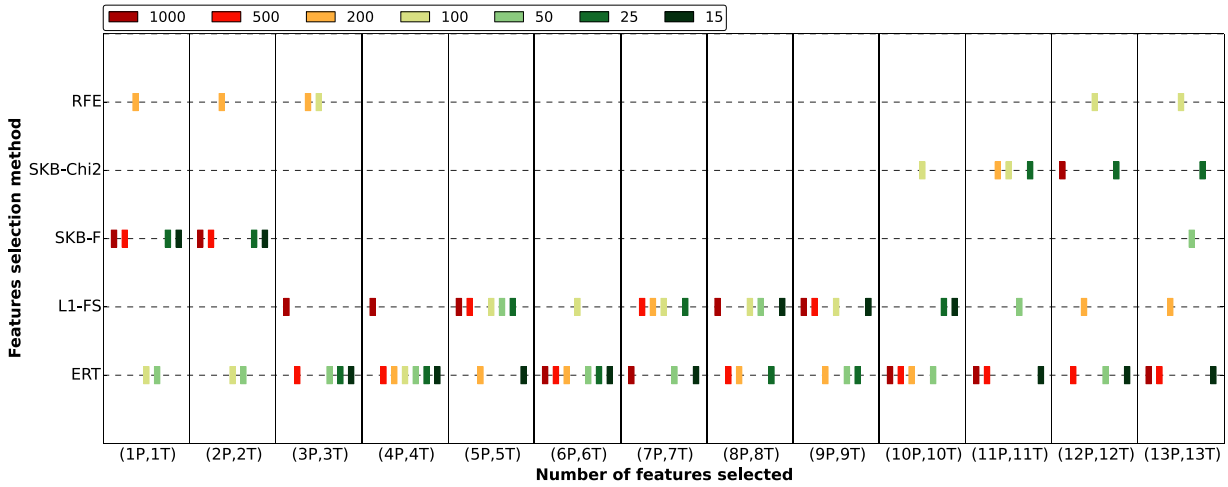


Fig. 7. Features selection methods adopted in order to select the best tuple of features. The (14P, 14T) tuple is not shown because all features are selected.

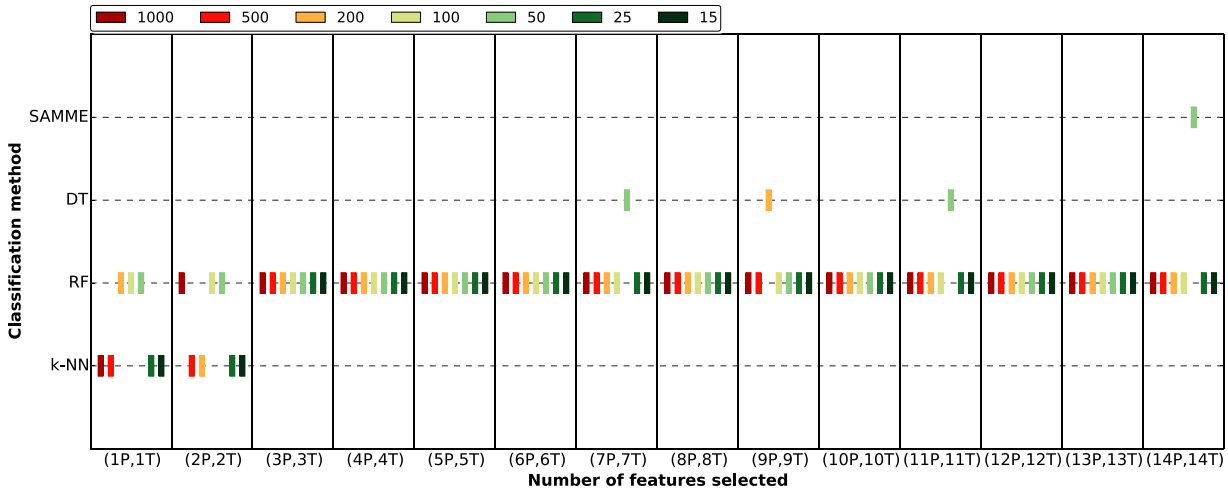


Fig. 8. Classification methods adopted in order to classify the best tuple of features.

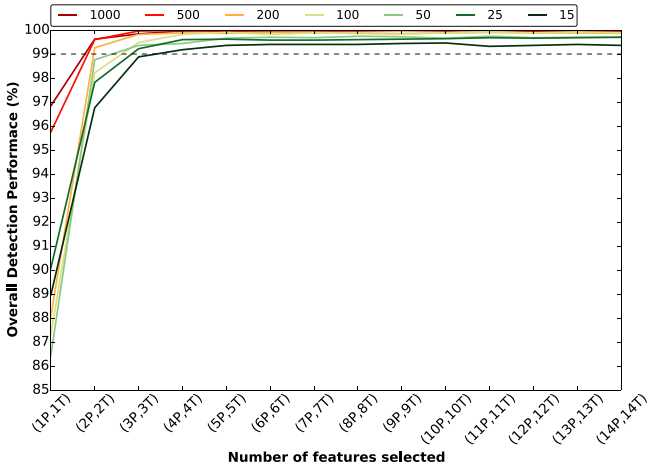


Fig. 9. ODP value related to best combination of features selection methods and classifiers that give the highest FES for different number of pings and features.

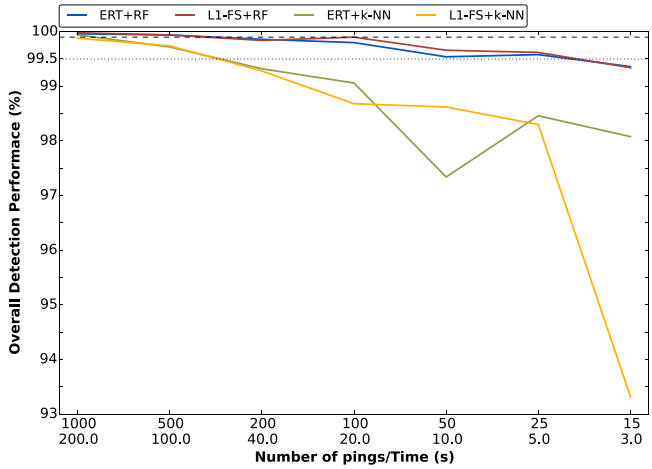


Fig. 10. ODP value related to best combination of features selection methods and classifiers for different number of pings and for the (5P, 5T) tuple.

characterization algorithm than the REM. Moreover, QEMU requires around 20% of the CPU of the Linux-based system. Therefore, with this system, a maximum of four VESs can be used at the same time without compromising its stability.

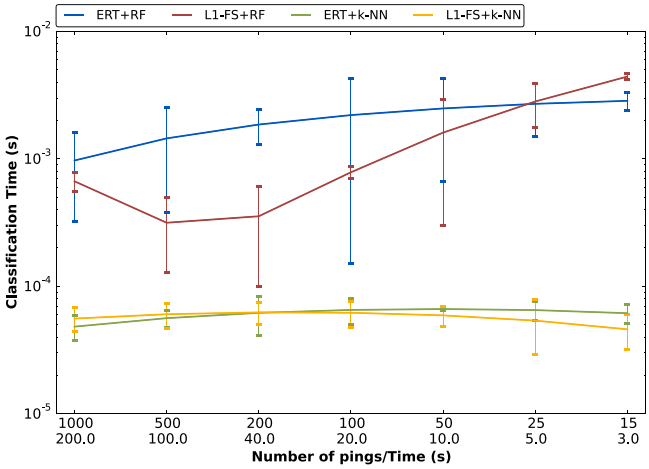


Fig. 11. Time required to classify a sample by using the best combination of features selection methods and classifiers for different number of pings and for the (5P, 5T) tuple.

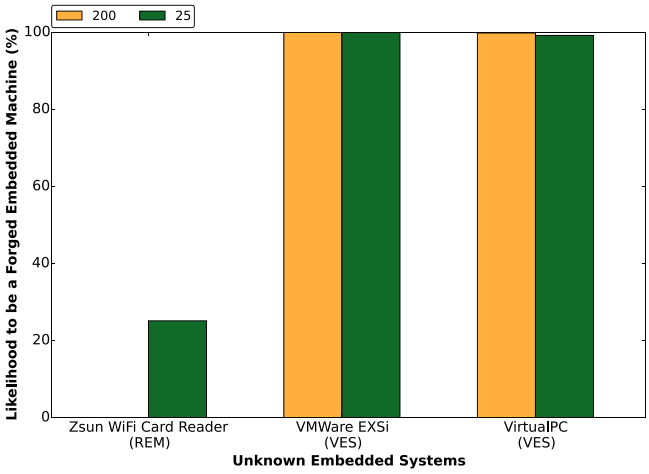


Fig. 12. Detection results by using the final detection method applied to UESs for 25 and 200 pings.

TABLE VI
EVALUATION OF THE CHARACTERIZATION ALGORITHM WITH 200 PINGS USING RASPBERRY PI AS REFERENCE SYSTEM

System	Total power consumption	Average CPU usage	Average memory usage
QEMU (Linux-based system)	45.59 W	20.26 %	16816.8 MB
VES	0.008 mW	0.09 %	0.256 MB
REM	0.001 mW	0.01 %	0.391 MB

VI. CONCLUSION

In this paper, we present a novel method for detecting forged embedded systems that has a high detection rate and a low detection time. A wide range of embedded machines was used to demonstrate its practical application. Behaviors of these embedded machines were used for creating a reference dataset in order to achieve reliable results. The results show that it can be employed to recognize forged embedded machines in IoT/M2M communications. Our method allows a quick trust assessment to be performed and can therefore identify forged embedded devices (potential attackers). In fact, by only adopting the trust TMFs proposed in previous works [12]–[18], real embedded machines will spend a lot of time trusting forged embedded machines by looking at their behaviors, reputations, relationships, etc., in the network. We demonstrated that it is possible to eliminate a possible threat in the network in only 5 s with a detection rate of more than 99.5%. Moreover, it is possible to increase the detection rate to 99.9% circa in only 40 s being also able to correctly detect the UESs. This detection mechanism can be used as pretrust evaluation by M2M embedded machines before agreeing to exchange information. These are very important aspects in the context of power-constrained machines, near real-time operations and dynamic networks, especially when applied to security. Importantly, the proposed method is also architecture and operating system independent. We are also able to show that the solution proposed can also be used to detect fake timing attacks (manuscript in preparation). Thanks to these features, it can be easily applied to existing and future IoT embedded machines.

Future research will involve the adoption of this trust evaluation as a preliminary step before evaluating the trust of machines in the network. This will avoid wasting energy and time to exchange information with other machines in the network in order to collect data for evaluating their trust. Moreover, other detection methods will be studied for embedded machines that do not support the ping functionality, like Bluetooth Low Energy machines.

REFERENCES

[1] Ericsson, Ericsson Mobility Report, 2015. [Online]. Available: http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility -report-nov-2015.pdf
[2] Gartner Inc., "Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015," 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317
[3] DHL Trend Research and Cisco Consulting Services, "Internet of Things in Logistics," 2015. [Online]. Available: http://www.dhl.com/content/dam/Local_Images/g0/New_aboutus/innovation/D HLTrendReport_Internet_of_things.pdf
[4] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.
[5] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson, "M2M: From mobile to embedded internet," *IEEE Commun. Mag.*, vol. 49, no. 4, pp. 36–43, Apr. 2011.
[6] G. M. Lee, N. Crespi, J. K. Choi, and M. Boussard, "Internet of things," in *Evolution of Telecommunication Services*. New York, NY, USA: Springer, 2013, pp. 257–282.
[7] J. A. Guerrero-ibanez, S. Zeadally, and J. Contreras-Castillo, "Integration challenges of intelligent transportation systems with connected vehicle, cloud computing, and internet of things technologies," *IEEE Wireless Commun.*, vol. 22, no. 6, pp. 122–128, Dec. 2015.
[8] F. Chiti, R. Fantacci, D. Marabissi, and A. Tani, "Performance evaluation of an efficient and reliable multicast power line communication system," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 7, pp. 1953–1964, Jul. 2016.
[9] W. Moreira and P. Mendes, "Pervasive data sharing as an enabler for mobile citizen sensing systems," *IEEE Commun. Mag.*, vol. 53, no. 10, pp. 164–170, Oct. 2015.
[10] M. Seufert, T. Griepentrog, V. Burger, and T. Hoßfeld, "A simple WiFi hotspot model for cities," *IEEE Commun. Lett.*, vol. 20, no. 2, pp. 384–387, Feb. 2016.
[11] P. K. Verma *et al.*, "Machine-to-machine (M2M) communications: A survey," *J. Netw. Comput. Appl.*, vol. 66, pp. 83–105, 2016.

[12] J. Guo, A. Marshall, and B. Zhou, "A new trust management framework for detecting malicious and selfish behaviour for mobile ad hoc networks," in *Proc. IEEE 10th Int. Conf. Trust, Security Privacy Comput. Commun.*, Nov. 2011, pp. 142–149.
[13] D. Chen, G. Chang, D. Sun, J. Li, J. Jia, and X. Wang, "TRM-IoT: A trust management model based on fuzzy reputation for internet of things," *Comput. Sci. Inform. Syst.*, vol. 8, no. 4, pp. 1207–1228, 2011.
[14] F. Bao and I.-R. Chen, "Dynamic trust management for internet of things applications," in *Proc. Int. Workshop Self-Aware Internet Things*, 2012, pp. 1–6.
[15] M. Nitti, R. Girau, L. Atzori, A. Iera, and G. Morabito, "A subjective model for trustworthiness evaluation in the social internet of things," in *Proc. IEEE 23rd Int. Symp. Personal Indoor Mobile Radio Commun.*, Sep. 2012, pp. 18–23.
[16] Y. B. Saied, A. Olivereau, D. Zeghlache, and M. Laurent, "Trust management system design for the internet of things: A context-aware and multi-service approach," *Comput. Security*, vol. 39, pp. 351–365, Nov. 2013.
[17] M. Nitti, R. Girau, and L. Atzori, "Trustworthiness management in the social internet of things," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 5, pp. 1253–1266, May 2014.
[18] Z. Chen, R. Ling, C.-M. Huang, and X. Zhu, "A scheme of access service recommendation for the social internet of things," *Int. J. Commun. Syst.*, vol. 29, pp. 694–706, 2015.
[19] V. Selis and A. Marshall, "MEDA: A machine emulation detection algorithm," in *Proc. 12th Int. Joint Conf. e-Business Telecommun.*, 2015, vol. 4, pp. 228–235.
[20] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection," in *Proc. 9th ACM Symp. Inf., Comput. Commun. Security*, 2014, pp. 447–458.
[21] Y. Jing, Z. Zhao, G.-J. Ahn, and H. Hu, "Morpheus: automatically generating heuristics to detect android emulators," in *Proc. 30th Annu. Comput. Security Appl. Conf.*, Dec. 2014, pp. 216–225.
[22] J. Rutkowska, "Red Pill: Detect VMM using (almost) One CPU Instruction," 2004. [Online]. Available: http://web.archive.org/web/20041130172213/http://invisiblethings.org/pa pers/redpill.html
[23] L. Martignoni, R. Paleari, G. F. Roglia, and D. Bruschi, "Testing CPU emulators," in *Proc. 18th Int. Symp. Softw. Testing Anal.*, 2009, pp. 261–272.
[24] H. Shi, A. Alwabel, and J. Mirkovic, "Cardinal pill testing of system virtual machines," in *Proc. 23rd USENIX Security Symp.*, 2014, pp. 271–285.
[25] T. Raffetseder, C. Kruegel, and E. Kirda, "Detecting system emulators," in *Information Security*. New York, NY, USA: Springer, 2007, pp. 1–18.
[26] W. Jia-Bin, L. Yi-Feng, and C. Kai, "Virtualization detection based on data fusion," in *Proc. Comput. Sci. Inf. Process.*, 2012, pp. 393–396.
[27] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 93–108, 2005.
[28] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *Proc. Dependable Syst. Netw. FTCS DCC*, 2008, pp. 177–186.
[29] L. Polčák and B. Franková, "On reliability of clock-skew-based remote computer identification," in *Proc. 11th Int. Conf. Security Cryptograph.*, 2014, pp. 291–298.
[30] L. Polčák, J. Jirásek, and P. Matousek, "Comment on "remote physical device fingerprinting," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 5, pp. 494–496, 2014.
[31] L. Polčák and B. Franková, "Clock-skew-based computer identification: Traps and pitfalls," *J. Universal Comput. Sci.*, vol. 21, no. 9, pp. 1210–1233, 2015.
[32] D. Quist and V. Smith, "Further down the VM spiral-detection of full and partial emulation for IA-32 virtual machines," *Proc. Defcon*, vol. 14, 2006, pp. 1–8.
[33] A. L. Ortega, *MAC Changer*, 2013. [Online]. Available: http://www.gnu.org/software/macchanger
[34] J. S. Robin and C. E. Irvine, "Analysis of the intel pentiums ability to support a secure virtual machine monitor," in *Proc. 9th Conf. USENIX Security Symp.*, 2000, vol. 9, p. 129.
[35] S. T. King and P. M. Chen, "Subvirt: Implementing malware with virtual machines," in *Proc. IEEE Symp. Security Privacy*, May 2006, pp. 3314–3327.
[36] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[37] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.
[38] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Mach. Learn.*, vol. 46, no. 1–3, pp. 389–422, 2002.
[39] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, 2008.
[40] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*, The Wadsworth Statistics/Probability series. Belmont, CA, USA: Wadsworth International Group, 1984, vol. 19.
[41] J. R. Quinlan, *C4.5: Programs for Machine Learning*, vol. 1, no. 3. Burlington, MA, USA: Morgan Kaufmann, 1993. [Online]. Available: http://portal.acm.org/citation.cfm?id=152181
[42] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, no. 2. New York, NY, USA: Springer, 2009, [Online]. Available: http://www.springerlink.com/index/D7X7KX6772HQ2135.pdf
[43] H. Zhang, "The optimality of Naive Bayes," *AA*, vol. 1, no. 2, p. 3, 2004.
[44] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost," *Statist. Interface*, vol. 2, no. 3, pp. 349–360, 2009.
[45] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[46] I. Guyon, B. Boser, and V. Vapnik, "Automatic capacity tuning of very large VC-dimension classifiers," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 5, 1993, pp. 147–155.
[47] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

Valerio Selis (M'13) was born in Cagliari, Sardinia, Italy, in 1983. He received the M.Sc. degree in computer science from the University of Cagliari, Cagliari, Italy. He is currently working toward the Ph.D. degree in electrical engineering and electronics at the University of Liverpool, Liverpool, U.K.

He was a Research Assistant with the Cross Layer Techniques for Intrusion Tolerant Network Design Project, Queen's University Belfast, Belfast, U.K. Since 2011, he has been a Development Engineer in wireless network security with Traffic Observation & Management Ltd. His research interests include wireless networks, Internet of Things, network security, trust management, and machine-to-machine communications.

Alan Marshall (M'88–SM'00) received the Ph.D. degree from the University of Aberdeen, Aberdeen, U.K., in 1991. He holds the Chair in Communications Networks with the University of Liverpool, where he is the Director of the Advanced Networks Research Group. He is also a Fellow of the IET with over 24 years working in the Telecommunications and Defense Industries. He has been a Visiting Professor in network security with the University of Nice/CNRS, France, and an Adjunct Professor for Research with Sunway University Malaysia. He has published more than 200 scientific papers and holds a number of joint patents in the areas of communications and network security. He has formed a successful spin-out company Traffic Observation & Management Ltd specializing in intrusion detection & prevention for wireless networks. His research interests include network architectures and protocols, mobile and wireless networks, network security, high-speed packet switching, quality of service & experience (QoS/QoE) architectures, and distributed haptics.