Marchal, Samuel; Miettinen, Markus; Nguyen, Thien Duc; Sadeghi, Ahmad-Reza; Asokan, N.

AuDI: Towards autonomous IoT device-type identification using periodic communications

Published in:
IEEE Journal on Selected Areas in Communications

DOI:
10.1109/JSAC.2019.2904364

Published: 01/06/2019

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Marchal, S., Miettinen, M., Nguyen, T. D., Sadeghi, A.-R., & Asokan, N. (2019). AuDI: Towards autonomous IoT device-type identification using periodic communications. *IEEE Journal on Selected Areas in Communications*, *37*(6), 1402-1412. Article 8664655. https://doi.org/10.1109/JSAC.2019.2904364

AUDI: Towards Autonomous IoT Device-Type Identification using Periodic Communication

Samuel Marchal, Markus Miettinen, Thien Duc Nguyen, Ahmad-Reza Sadeghi, N. Asokan

***Abstract*—IoT devices are being widely deployed. But the huge variance among them in the level of security and requirements for network resources makes it unfeasible to manage IoT networks using a common generic policy. One solution to this challenge is to define policies for classes of devices based on *device type*.**

In this paper, we present AUDI, a system for quickly and effectively identifying the type of a device in an IoT network by analyzing their network communications. AUDI models the periodic communication traffic of IoT devices using an unsupervised learning method to perform identification. In contrast to prior work, AUDI operates *autonomously* after initial setup, learning, without human intervention nor labeled data, to identify previously unseen device types. AUDI can identify the type of a device in *any mode of operation or stage of lifecycle* of the device. Via systematic experiments using 33 off-the-shelf IoT devices, we show that AUDI is effective (98.2% accuracy).

***Index Terms*—Internet of Things, device-type identification, autonomous IoT device identification, self-learning**

I. INTRODUCTION

The growing popularity of the *Internet-of-Things* (IoT) has led to widespread use of IoT devices, especially in *Small Office and Home* (SOHO) settings where appliances and devices are increasingly connected to the Internet. While IoT enables useful functionality like remote monitoring and control, it transforms the structure and nature of typical SOHO networks, raising new challenges in network management.

IoT devices can be inherently mobile and hence dynamic. They are also more *heterogeneous* compared to general-purpose computing devices. Different IoT devices have different Quality of Service (QoS) requirements like network bandwidth or tolerance to packet loss. For instance, a connected camera requires higher bandwidth (when streaming video) than a smart light bulb. A connected smoke detector or a smart key lock requires more reliable communications in contrast to a smart coffee maker. Failed message delivery can endanger lives (in the case of the smoke detector) or threaten security (key lock). Security of commodity IoT devices is often dismal. Devices are routinely released to the market with easily exploitable security vulnerabilities [1]–[3]. The increased heterogenity and weak security significantly raise the difficulty of managing IoT networks.

One way to make these challenges tractable stems from the observation that QoS and security requirements of IoT devices of the same *type* tend to be similar. For example, high end IP cameras from the same device manufacturer are likely to have similar bandwidth requirements. Similarly, devices from a given manufacturer running the same version of firmware will have the same vulnerabilities (e.g., enabling debug mode will open a backdoor [4]). QoS and security policies can thus be made more manageable by specifying them in terms of *device types*. Previous work used device fingerprinting approaches to identify the device model [5]–[7] and/or the specific hardware/software configuration of a device [5], [8]–[11] by training classification models with *labeled data* from specific known device types. Such training data requires extensive human effort to generate and maintain which is particularly difficult given the increasing number and variety of IoT device manufacturers.

We take a different approach in this paper: the purpose of device identification is to enable automated network management. Hence, there is no need to identify the actual real-world model of a device. It is sufficient to reliably map devices to an *abstract "device type"* for which the system has learned a specific set of policies (QoS or security). Therefore, such a system can be trained without the need to manually label the communication traces of predefined real-world device types.

Goals and Contributions. Our goal is to develop a technique for quickly, accurately and *autonomously* identifying the type of IoT devices connected to a SOHO network. Autonomy in this context refers to minimizing reliance on human assistance such as requiring substantial labeled ground truth data for training. It must scale to the large number of IoT devices already available on the market and quickly adapt to new IoT devices appearing on the market. Furthermore, our solution must be capable of detecting device types in *any operational mode* (standby or user interaction) and at *any stage of the lifecycle of a device* so that the solution can be introduced into an *existing IoT network*. This is in contrast to previous solutions that can only detect device type when a device is first introduced to the network [5], or during active usage of the device [5]–[7]. Major IoT device vendors, including Cisco, helped us formulate the setting for our solution and potential usage scenarios.

We make the following contributions:

- AUDI, an autonomous, distributed system for learning and identifying the type of an IoT device (Sect. II).
- a novel device-type identification method (Sect. IV) based on passive fingerprinting of periodic communication traffic of IoT devices (Sect. III). Unlike previous methods, it requires *no prior knowledge* of device types nor labeled training data and is effective at identifying the type of an IoT device in *any operation mode of a device*.

Samuel Marchal and N. Asokan are with Aalto University (Finland) - emails: samuel.marchal@aalto.fi, asokan@acm.org.

Markus Miettinen, Thien Duc Nguyen and Ahmad-Reza Sadeghi are with Technische Universitat Darmstadt (Germany) - emails: {markus.miettinen,ducthien.nguyen,ahmad.sadeghi}@trust.tu-darmstadt.de

- experimental evaluation of AuDI using a large dataset comprising 33 typical commercial IoT devices (Sect. V) showing that AuDI achieves 98.2% accuracy. We will make our datasets as well as the AuDI implementation available for research use.

II. System Model

A. Model and requirements

We target a system model (Fig. 1) of a typical SOHO network where IoT devices connect to the Internet via an access gateway. The primary goal of AuDI is to enable the gateway to identify the type of devices connected to it. The identification relies on passively monitoring network communications from connected devices. AuDI must meet the following requirements.

R1 **Autonomy.** Operate with limited human intervention, without requiring labeled training data.

R2 **Scalability.** Be able to manage a large number of device types and learn to identify new types as they emerge.

R3 **Stability.** Be able to function consistently effectively (speed and accuracy) regardless of the lifecycle stage (induction vs. normal operation) or operational mode (standby or user interaction) of the target IoT device.

R4 **Security.** Be resilient to *spoofing attacks* by an adversary modifying communications of a compromised device, trying to masquerade as a different device type.

B. AuDI *system design*

AuDI consists of an *IoT Cloud Service* and several *IoT Gateways*.

IoT Gateway acts as the local access gateway to the Internet. IoT devices connect to it, e.g., over WiFi or ethernet. Apart from acting as a gateway router for connected devices in the local network, *IoT Gateway* hosts the AuDI *Device Fingerprinting* component which monitors the communication patterns of connected IoT devices to extract *device fingerprints* (details in Sect. III and IV).

IoT Cloud Service is a cloud-based functionality hosting the AuDI *Device-Type Identification* component which uses a machine learning-based classifier for *identifying the device type* of IoT devices based on device fingerprints provided by *IoT Gateway*. *IoT Cloud Service* aggregates fingerprints from several *IoT Gateways* to learn device-type identification models (details in Sect. IV-B). Aggregation helps overall accuracy because the amount of IoT communication observed at any given *IoT Gateway* is likely to be small.

AuDI automatically generates and assigns *abstract labels* to represent individual device types. This is done by first building fingerprints for the communication patterns of each IoT device. Then, AuDI uses an unsupervised clustering algorithm to autonomously group these fingerprints into clusters and create an abstract label for each cluster (details in Sect. IV-B). The whole process does not require any human intervention. It is worth noting that AuDI starts operating with no device-type identification model. It learns and improves the device-type identification model as *IoT Gateways* aggregate more data.

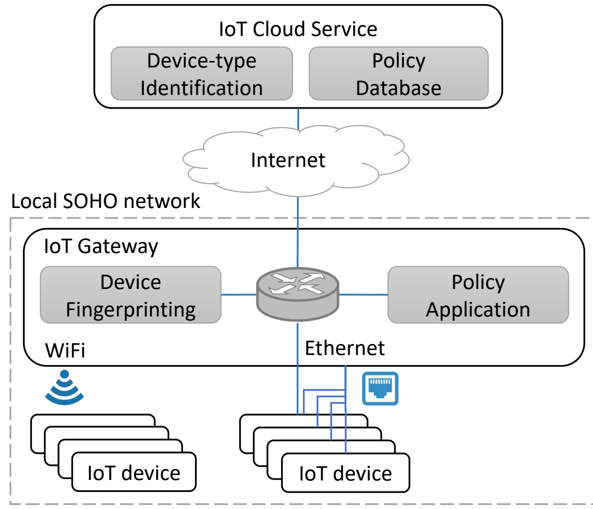

Fig. 1: AuDI system model

C. Device-type-specific policies

Device type identification provided by AuDI can be used in a variety of ways. We briefly outline the architectural approach and some possible uses here but refer the reader to our concurrent work [12] for a detailed description. Figure 1 illustrates the architecture to facilitate device-type-specific policies to enhance IoT network management. First, we augment *IoT Cloud Service* with a *Policy Database* that associates a set of policies with each detected device type. When an *IoT Gateway* detects a device of a certain type, it can retrieve the corresponding policy from *Policy Database*. Policies can be centrally formulated (e.g., by experts) and/or locally learned on each *IoT Gateway* and aggregated in *Policy Database*. Some example applications for device-type-specific policies are:

- **Anomaly detection**: By monitoring devices of a given type, *IoT Gateway* can learn a profile for their normal behavior. Behavior profiles learned locally can be aggregated into a global *device-type-specific anomaly detection profile*. When *IoT Gateway* detects a new device of a certain type, it can retrieve the anomaly detection profile for that type from *IoT Cloud Service* and use it right away to detect any anomalous behavior involving that device. This scenario is described in detail in [12].
- **Network resource allocation**: By monitoring the communication of a specific device type over time, *IoT Gateway* can learn its requirements in terms of, e.g., network bandwidth. Again, requirements learned from different *IoT Gateways* can be aggregated so that a new *IoT Gateway* can provision resources to a newly detected device without delay.
- **Identification and isolation of vulnerable IoT devices**: A device type can be linked to known vulnerabilities by using vulnerability reports, as proposed in [5]. Once *IoT Gateway* detects a device whose type is known to have vulnerabilities, it can enforce a policy for constraining the communications of the device [5] using Software-Defined Networking techniques [13]. Such policies are specified

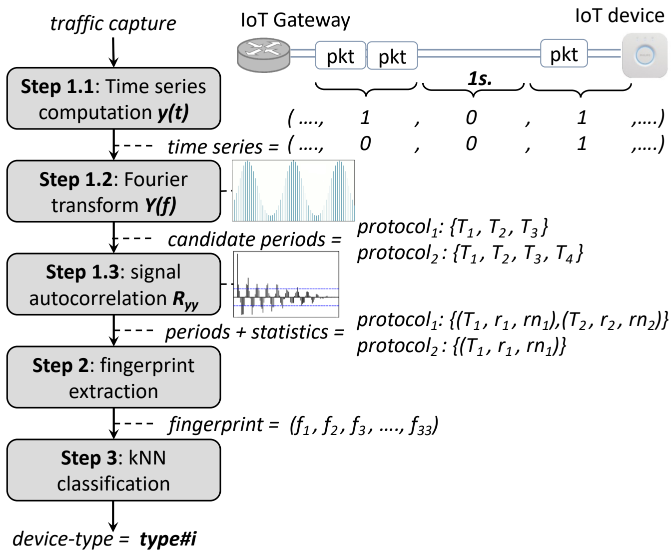

Fig. 2: Overview of device-type identification.

in *IoT Cloud Service Policy Database*.

D. Device-type identification overview

Traditional device-type identification approaches [6], [7] rely on aggregated statistics extracted from dense network traffic. These are ineffective when applied to IoT devices due to the scarcity of their communication. IoT devices generate little dense traffic, typically only during rare and short user interactions. Nevertheless, IoT devices also generate background communication independent of user interactions. This traffic is always present, relatively constant and periodic.

Thus, we introduce a novel technique for identifying the type of IoT devices based on their periodic background network traffic. In contrast to existing approaches, this technique can identify the type of an IoT device *in any state of a device's operation*, including standby, with a constant time of 30 minutes. Our technique is composed of three steps relying on passive monitoring of the network traffic at the network gateway: step 1: inference of periodic flows, their period and stability (Sect. III), step 2: extraction of a fingerprint characterizing a device's type based on its periodic flows (Sect. IV-A) and step 3: use of this fingerprint in a classification system that identifies device-types (Sect. IV-B). The overview of the identification process is depicted in Fig. 2. Steps 1 and 2 are implemented in *IoT Gateway* while step 3 is implemented in *IoT Cloud Service*.

III. Periodic Flow Inference

Fourier transform and signal autocorrelation are effective signal processing techniques for inferring periodicity. While Fourier transform and signal autocorrelation can identify the several distinct periods of a signal, ignoring most non-periodic noise, these techniques are more accurate when applied to pure single-periodic signals. As a result, we pre-process the network traffic received at *IoT Gateway* and divide it into distinct *flows*. We define a *flow* as a sequence of network packets sent from a given source MAC address (IoT device) using a given communication protocol (*e.g.*, NTP, ARP, RTSP, etc.). The rationale for flow division is that most periodic communication uses dedicated protocols that are different from the ones used for communication related to user interaction (non-periodic). If periodic and non-periodic communication still coexist in a flow (e.g., HTTP), Fourier Transform and signal autocorrelation can cope better with this reduced non-periodic noise.

The flow of packets in a network capture must be converted into a format suitable for signal processing. We discretize each flow into a binary time series sampled at one value per second, indicating whether the flow contained one or more packets during the 1-second period (value 1) or not (value 0). The computed time series is a discrete binary signal $y(t)$ of duration d seconds.

We first use the discrete Fourier transform (DFT) [14] to identify candidate periods for a given flow. DFT converts a discrete signal $y(t)$ from the time domain to the frequency domain: $y(t) \Rightarrow Y(f)$. $Y(f)$ provides amplitude values for each frequency $f \in [0; 1]$. The frequency f_i resulting in the largest amplitude $Y(f_i) = max(Y(f))$ gives the periodicity $T_i = \frac{d}{f_i}$ of the dominant period in $y(t)$. Secondary periods T_j of lower amplitude also exist. We select candidate periods T_i having an amplitude $Y(\frac{d}{T_i})$ larger than 10% of the maximum amplitude $max(Y(f))$. We discard close candidate periods by selecting only local maxima of Y. $Y(f)$ is considered a local maximum on Y if $Y(f-1) < Y(f) > Y(f+1)$. The result of this operation is a list of candidate periods for a flow.

Candidate periods found using DFT can be nonexistent or inaccurate. To confirm and refine these periods, we compute the discrete autocorrelation R_{yy} of $y(t)$. R_{yy} denotes the similarity of the signal $y(t)$ with itself as a function of different time offsets. If R_{yy} at offset l is large and reaches a local maximum, it means that $y(t)$ is likely periodic, with period $T = l$ and that this period occurs $R_{yy}(l)$ times over $y(t)$. For each candidate period T_i obtained with DFT, we confirm and refine it by analyzing the value of $R_{yy}(l_i)$ on the range of close offsets $l_i \in [0.9 \times T_i; 1.1 \times T_i]$. If it contains a local maximum $R_{yy}(l_i) = lmax_i$, we confirm the existence of a period that belongs to this range and update its value to $T_i = l_i$. $R_{yy}(l_i)$ is considered a local maximum on R_{yy} if $R_{yy}(l_i - 1) < R_{yy}(l_i) > R_{yy}(l_i + 1)$. For each resulting period T_i we compute characteristic metrics r_i and rn_i, defined as:

$$r_i = \frac{T_i \times R_{yy}(T_i)}{d} \tag{1}$$

$$rn_i = \frac{T_i \times (R_{yy}(T_i - 1) + R_{yy}(T_i) + R_{yy}(T_i + 1))}{d} \tag{2}$$

r_i computes the ratio of occurrences of period T_i over signal $y(t)$ of duration d seconds. An accurate and stable periodic signal of period T_i renders $r_i = 1$. However, a periodic signal may be noisy ($r_i < 1$) or have parallel periods with the same periodicity. Periodic signals may also be unstable exhibiting slight differences in their periodicity ($r_i < 1$). This is the rationale for computing rn where we sum the occurrences of neighboring periods $R_{yy}(T_i - 1)$, $R_{yy}(T_i)$ and $R_{yy}(T_i + 1)$. A

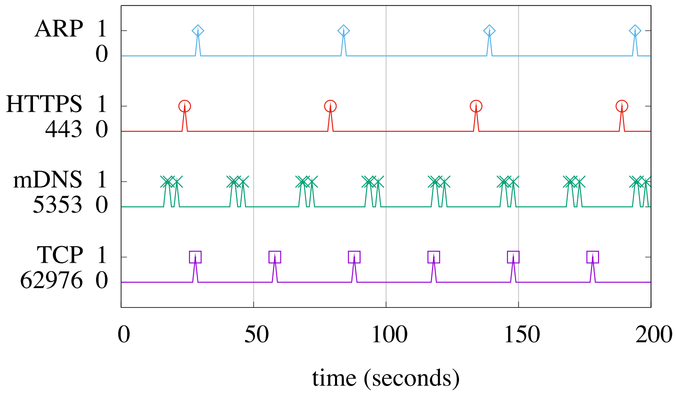

Fig. 3: Four binary time series extracted from periodic flows of a D-LinkCam DCS935L IP camera. The flows correspond to ARP protocol, HTTPS (port 443), mDNS (port 5353) and TCP port 62976. All flows are periodic.

stable signal of period T_i produces $r_i \approx rn_i \approx 1$, while unstable signals produce $r_i < 1$ and $r_i \ll rn_i$.

The final result of period inference for a flow is a set of periods with the corresponding ratios r_i and rn_i: $\{(T_1, r_1, rn_1), \cdots, (T_i, r_i, rn_i), \cdots, (T_n, r_n, rn_n)\}$.

Example: Figure 3 shows the plot of binary time series extracted from flows of a D-LinkCam DCS935L IP camera. We see that all depicted flows are periodic. Applying DFT and autocorrelation on these time series provides the following results:

```
ARP: (period:55, r:0.735, rn:0.857)
HTTPS: (period:55, r:0.857, rn:1.102)
mDNS: (period:25, r:2.171, rn:4.399)
port 62976: (period:30, r:0.969, rn:0.969)
```

We see that our method is able to accurately infer all periods observed in Fig. 3. The flow on TCP port 62976 has the most stable period (30s.) as highlighted by the values $r = rn \approx 1$. ARP and HTTPS (port 443) have both a less stable period of 55s., as highlighted by lower r values and a larger difference between r and rn. We also inferred the 25s. period of the mDNS flow (port 5353). But as we can observe in Fig. 3, there are three different signals having a 25s. periodicity on this flow. This aspect is captured in period inference by rendering high r and rn values, i.e. far larger than 1. These results show that our method detects periodic flows, accurately infers their period and characterizes them with r and rn metrics.

IV. Device-Type Identification

We build a fingerprint for a device type by extracting features from its periodic flows. These features are later used with an unsupervised machine learning algorithm that creates and assigns device-type labels to fingerprints.

A. *Fingerprint extraction*

We split a network traffic capture of x seconds into three sub-captures $[0; \frac{x}{2}], [\frac{x}{4}, \frac{3x}{4}], [\frac{x}{2}, x]$. We apply periodic flow inference (Sect. III) on each sub-capture and on the whole capture $[0; x]$. We obtain four sets of periods with the metrics r and rn for each flow. The goal of applying period inference on smaller sub-captures is twofold. First, we obtain more significant results by discarding periods that are inferred from less than two sub-captures. Second, we can compute statistics from metrics r and rn to measure their stability.

The results from period inference are grouped by source MAC address, linked to a single device. This grouping defines the granularity of feature extraction, i.e., one fingerprint is extracted per source MAC address and capture. We introduce 33 features that compose our device-type fingerprint. These features are manually designed to model a group of periodic flows in a unique manner that enables to distinguish device types. It is worth noting that all our features are computed from the statistics obtained during periodic flow inference (Sect. III). They do not use packet payload information nor packet header information from protocols above the transport layer. Consequently, AuDI can operate on any traffic encrypted above the transport layer. There are four categories of features as discussed below and in Tab. I.

Periodic flows (9 features). This feature category characterizes the quantity and quality of periodic flows. It includes the count of periodic flows (1), the layer of protocols that support periodic flows (2), if flows are single- or multi-periodic (3-6), if there is a change in the source port of periodic flows (7) and the frequency of this change (8-9).

Period accuracy (3 features). These features measure the accuracy of the inferred periods and characterises how noisy the flows they were extracted from are. They consist of the count of periods that were inferred from all sub-captures and the whole capture (10), the mean (11) and standard deviation (12) for the count of sub-captures from which each period was inferred.

Period duration (4 features). These features (13-16) represent the number of periods that belong to four duration ranges, e.g., $[5s.; 29s.]$. The ranges were manually chosen in an attempt to segregate periods according to their relative duration: $[5s.; 29s.]; [30s.; 59s.]; [60s.; 119s.]; [120s.; 600s.]$. Periods of less than 5 seconds or more than 10 minutes are discarded. Identifying long periods requires long traffic captures which slows down the fingerprint extraction.

Period stability (17 features). Features in this category measure the stability of the inferred periods using r and rn metrics, as discussed in Sect III. The mean and standard deviation (SD) of r and rn metrics are computed for each flow and period. Features 17-20, respectively 24-27, are calculated by binning the values of Mean(r), respectively Mean(rn), into four ranges and counting the number of values in each bin. The bin ranges of mean r and rn values were selected to distinguish noisy $[0.2; 0.7[$ from pure $[0.7; 1[$ single-period flows as well as different multi-periodic flows $[1; 2[$, $[2; +\infty[$. Features 21-23, respectively 28-30, are calculated by binning the values of SD(r), respectively SD(rn), into three ranges and counting the number of values in each bin. These ranges were selected to distinguish very stable $[0; 0.02[$ from stable $[0.02; 0.1[$ and unstable $[0.1; +\infty[$ periodic flows. Features 31-33 are computed by binning the values of the difference $rn - r$ and into three ranges of values and counting the corresponding bin cardinalities. These ranges were selected to characterize the differences between stable and unstable periods of flows.

TABLE I: 33 features (4 categories) used for device-type identification. # represents a count, SD is the standard deviation. Importance scores are computed using ReliefF feature selection algorithm [15]. High scores (green) corresponds to the most relevant features and low scores (red) to the least relevant features.

Category	f	Description	Importance
periodic flows	1	# periodic flows	0.440
	2	# periodic flows (protocol $\leq$ layer 4)	0.465
	3	Mean periods per flow	0.068
	4	SD periods per flow	0.037
	5	# flows having only one period	0.429
	6	# flows having multiple periods	0.176
	7	# flows with static source port	0.533
	8	Mean frequency source port change	0.310
	9	SD frequency source port change	0.137
period accuracy	10	# periods inferred in all sub-captures	0.329
	11	Mean period inference success	0.037
	12	SD period inference success	0.022
period duration	13	# periods $\in [5s.; 29s.]$	0.409
	14	# periods $\in [30s.; 59s.]$	0.408
	15	# periods $\in [60s.; 119s.]$	0.467
	16	# periods $\in [120s.; 600s.]$	0.419
period stability	17	# Mean(r) $\in [0.2; 0.7[$	0.386
	18	# Mean(r) $\in [0.7; 1[$	0.436
	19	# Mean(r) $\in [1; 2[$	0.239
	20	# Mean(r) $\in [2; +\infty[$	0.124
	21	# SD(r) $\in [0; 0.02[$	0.185
	22	# SD(r) $\in [0.02; 0.1[$	0.151
	23	# SD(r) $\in [0.1; +\infty[$	0.185
	24	# Mean(rn) $\in [0.2; 0.7[$	0.288
	25	# Mean(rn) $\in [0.7; 1[$	0.307
	26	# Mean(rn) $\in [1; 2[$	0.313
	27	# Mean(rn) $\in [2; +\infty[$	0.246
	28	# SD(rn) $\in [0; 0.02[$	0.217
	29	# SD(rn) $\in [0.02; 0.1[$	0.217
	30	# SD(rn) $\in [0.1; +\infty[$	0.220
	31	# Mean(rn)− Mean(r) $\in [0; 0.02[$	0.408
	32	# Mean(rn)− Mean(r) $\in [0.02; 0.1[$	0.248
	33	# Mean(rn)− Mean(r) $\in [0.1; +\infty[$	0.482

B. Device-type fingerprint classification

Our device-type identification technique is designed to be fully autonomous. It does not require human interaction nor labeled data to operate. When an IoT device is associated to an *IoT Gateway*, the latter monitors its network traffic and extracts a fingerprint as described in Sect. IV-A. The fingerprint is sent to the *IoT Cloud Service*, which attempts to identify the type of the device having this fingerprint. If the fingerprint has a match, the type of the device is identified and the fingerprint is used to retrain and improve its identification model. If no match is found, the *IoT Cloud Service* uses the fingerprints to learn a model for this new device type.

AUDI starts operating with no identification model. As *IoT Cloud Service* receives fingerprints from *IoT Gateway*, it creates type identifiers (e.g., *type#12*) and learns an identification model for them. The longer the system runs and the more *IoT Gateways* contribute to it, the more device types it is able to identify and the better the accuracy of identification.

We implement automated device-type identification using a supervised k-Nearest Neighbors (kNN) classifier [16]. kNN is chosen because of its ability to deal with a large number of classes and an imbalanced dataset. Each device type is represented by one class and the training data available for each class may be imbalanced (as IoT devices are differently deployed). kNN forms small clusters of at least k neighbors to represent a class. In a supervised mode, several clusters can define a class, capturing its potential diversity. This allows fingerprints collected from a device from which we already know the type to form new clusters with the same type label. When fingerprints for device types unknown to the model are processed, they are detected as exceeding a threshold distance to the nearest cluster of the classification model. A new class can be added to the model to represent this yet unknown device type.

Our features are processed and should not require complex association to differentiate device types. Consequently, we use the Euclidian distance as distance measure in kNN. All 33 features of our fingerprints are scaled on the range $[0; 1]$ to have an equal weight in the classification task. Fingerprints are extracted from network traffic captures of 30 minutes. We tested several capture durations: $\{5, 10, 20, 30, 60, 90\}$ minutes. A duration lower than 30 minutes missed flows of long periodicity (10 minutes) and degraded the accuracy of identification. A duration longer than 30 minutes did not improve accuracy but increased the delay to identify a device. We set $k = 5$ to meet a trade-off between representativeness of a learned class and need for training data. A class for a new device-type can be learned as soon as we get five fingerprints for it, i.e., after 2.5 hours of monitoring.

The design of our fingerprint classification approach does not require any labeled data to operate. It allows AUDI to learn and label device types without human intervention by clustering fingerprints and generating labels for clusters. Four parameters need to be tuned and defined prior to deployment of AUDI: the traffic capture duration, the sampling period of the flows, k and the threshold distance for kNN. Optimal values for these parameters can be determined in a lab setup using a small set of IoT devices. After that, AUDI can run in a fully autonomous manner, without human intervention. Optimal parameter values inferred from a constrained lab setup may not generalize well to larger deployments. These can be later adjusted in real-time by trying to re-identify already identified IoT device types as if they were unknown devices (in a supervised learning fashion). Several parameter values can then be tested automatically using a grid search strategy or Bayesian optimization [17]. The set of parameter values achieving the best accuracy and speed of identification can be selected and applied to AUDI. Consequently, AUDI meets the autonomy requirement **R1** by design. Our approach allows AUDI to manage a large number of device types since these are represented as clusters in a high dimensional space (33 dimensions). A multitude of non overlapping clusters can be created in this space. The addition of new device types to the system is an automatic process of creating new clusters in this space. As a result we can conclude that AUDI also meets **R2**.

TABLE II: 33 IoT devices used in the *background* and *activity* datasets and their connectivity technologies + Affectation of these devices to 23 AuDI device types during evaluation.

Device-type	Identifier	Device model	WiFi	Ethernet	Other	Background	Activity
type#01	ApexisCam	Apexis IP Camera APM-J011	●	●	○	●	●
type#02	CamHi	Cooau Megapixel IP Camera	●	●	○	●	●
type#03	D-LinkCamDCH935L	D-Link HD IP Camera DCH-935L	●	○	○	●	●
type#04	D-LinkCamDCS930L	D-Link WiFi Day Camera DCS-930L	●	●	○	●	○
	D-LinkCamDCS932L	D-Link WiFi Camera DCS-932L	●	●	○	●	○
type#05	D-LinkDoorSensor	D-Link Door & Window sensor	○	○	●	●	●
	D-LinkSensor	D-Link WiFi Motion sensor DCH-S150	●	○	○	●	●
	D-LinkSiren	D-Link Siren DCH-S220	●	○	○	●	●
	D-LinkSwitch	D-Link Smart plug DSP-W215	●	○	○	●	●
	D-LinkWaterSensor	D-Link Water sensor DCH-S160	●	○	○	●	●
type#06	EdimaxCamIC3115	Edimax IC-3115W Smart HD WiFi Network Camera	●	●	○	●	●
	EdimaxCamIC3115(2)	Edimax IC-3115W Smart HD WiFi Network Camera	●	●	○	●	●
type#07	EdimaxPlug1101W	Edimax SP-1101W Smart Plug Switch	●	○	○	●	●
	EdimaxPlug2101W	Edimax SP-2101W Smart Plug Switch	●	○	○	●	●
type#08	EdnetCam	Ednet Wireless indoor IP camera Cube	●	●	○	●	●
type#09	EdnetGateway	Ednet.living Starter kit power Gateway	●	○	●	●	●
type#10	HomeMaticPlug	Homematic pluggable switch HMIP-PS	○	○	●	●	●
type#11	Lightify	Osram Lightify Gateway	●	○	●	●	●
type#12	SmcRouter	SMC router SMCWBR14S-N4 EU	●	●	○	●	●
type#13	TP-LinkPlugHS100	TP-Link WiFi Smart plug HS100	●	○	○	●	●
	TP-LinkPlugHS110	TP-Link WiFi Smart plug HS110	●	○	○	●	●
type#14	UbnTAirRouter	Ubnt airRouter HP	●	●	○	●	●
type#15	WansviewCam	Wansview 720p HD Wireless IP Camera K2	●	○	○	●	●
type#16	WeMoLink	WeMo Link Lighting Bridge model F7C031vf	●	○	●	●	●
type#17	WeMoInsightSwitch	WeMo Insight Switch model F7C029de	●	○	○	●	●
	WeMoSwitch	WeMo Switch model F7C027de	●	○	○	●	●
type#18	HueSwitch	Philips Hue Light Switch PTM 215Z	○	○	●	●	●
type#19	AmazonEcho	Amazon Echo	●	○	○	○	●
type#20	AmazonEchoDot	Amazon Echo Dot	●	○	○	○	●
type#21	GoogleHome	Google Home	●	○	○	●	○
type#22	Netatmo	Netatmo weather station with wind gauge	●	○	●	●	○
type#23	iKettle2	Smarter iKettle 2.0 water kettle SMK20-EU	●	○	○	●	●
	SmarterCoffee	Smarter SmarterCoffee coffee machine SMC10-EU	●	○	○	●	●

V. Evaluation

A. Datasets

To evaluate AuDI, we collected extensive datasets of communication traces of IoT devices in a laboratory setting which consisted of 33 typical consumer IoT devices like IP cameras, smart power plugs and light bulbs, sensors, etc. (See Tab. II for the full list).

1) Description: **Background dataset.** Our first dataset is representative of background traffic IoT devices generate when no explicit user actions are involved . It captures communications resulting from standby mode operations such as heartbeat messages, regular status updates or notifications.

Activity dataset. Our second dataset is representative of traffic triggered by user activity. A key characteristic of IoT devices is that they are typically single-purpose devices with inherently less diverse behavior than general-purpose computing devices like desktop computers, or smartphones. Most devices expose only a few distinct actions to users, e.g., ON, OFF, ADJUST, etc. This dataset encompasses all such actions being invoked on the respective IoT devices, thus capturing the full diversity of behavior related to user activity,.

2) Data Collection: Our data collection setup in the laboratory network is shown in Fig. 4. We used `hostapd` on a laptop running Kali Linux to create an *IoT Gateway* acting as an access point with WiFi and Ethernet interfaces to which all IoT devices were connected. On the *IoT Gateway* we used *tcpdump* to collect network traffic packets, filtering out packets not related to a given device based on the device's MAC address. Most of the devices used either WiFi or Ethernet. Some devices like smart light bulbs or sensors, used low-energy protocols like ZigBee, Z-Wave or Bluetooth Low Energy (BLE) to connect to a hub device which was then connected over WiFi or ethernet to the network. For these devices, we therefore monitored indirect traffic between the hub device and our *IoT Gateway*.

Background dataset. Background traffic was collected configuring the devices to the laboratory network, verifying the correctness of the setup and then leaving the devices on their own for a period of at least 24 hours. During this time, no user interactions with the devices were done.

Activity dataset. Activity data was collected by connecting each IoT device to the laboratory network and repeatedly performing actions shown in Tab. III with the devices. Each action was repeated 20 times, while leaving short pauses of random duration between individual actions. To also capture less intensive usage patterns, the dataset was augmented with longer measurements of two to three hours, during which actions were triggered only occasionally. The activity dataset contains data from 27 IoT devices, a subset of those used for collecting the background dataset, as some devices like the

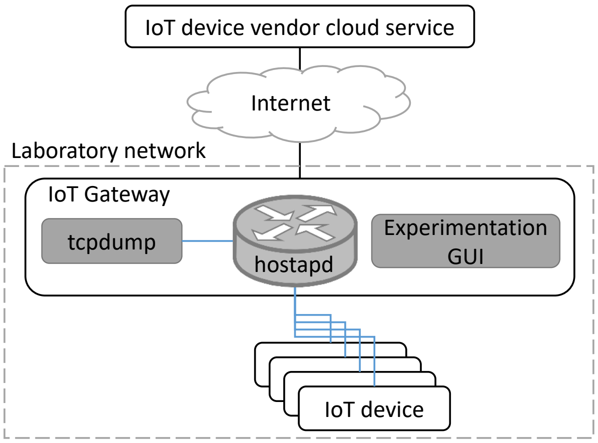

Fig. 4: Lab network setup

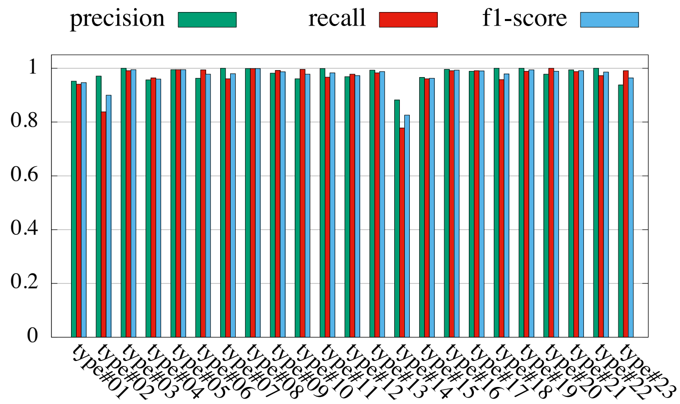

Fig. 5: Precision, recall and f1-score for identification of 23 device types (e.g., *type#01*).

TABLE III: Actions for different IoT device categories

Category (count)	Typical actions
IP cameras (6)	START / STOP video, adjust settings, reboot
Smart plugs (9)	ON, OFF, meter reading
Sensors (3)	trigger sensing action
Smart lights (4)	turn ON, turn OFF, adjust brightness
Actuators (1)	turn ON, turn OFF
Appliances (2)	turn ON, turn OFF, adjust settings
Routers (2)	browse amazon.com

above-mentioned hub devices for lighting or home automation do not allow any meaningful user interaction.

Table III shows the typical actions invoked for different types of IoT devices. Table IV summarizes the sizes and numbers of distinct packets and packet flows contained in the two datasets. While packet flows cannot be directly mapped to distinct device actions, they do, however, provide a rough estimate of the device's overall activity.

B. Device-type identification: accuracy and speed

To evaluate the accuracy of our device-type identification technique, we computed fingerprints (cf. Sect. IV-A) from the *background* and *activity* traffic dataset. We obtained 6,224 fingerprints representing 33 IoT devices.

To assess the relevance of our automatically defined device types, we trained a kNN model from the fingerprints, following the method presented in Sect. IV-B. It defined 23 classes (device types). 16 devices were each assigned its own separate device type. The remaining 17 were aggregated into 7 device types. The assignment of devices to automatically-defined device types is summarized in Tab. II. Different devices allotted to a given device type are always from the same manufacturer and have the same or similar purpose (smart plugs / IP cameras / smart switches / sensors). For example,

TABLE IV: Characteristics of used datasets

Dataset	Size (MB)	Flows	Packets
Background	226	56,337	127,532
Activity	239	59,577	134,867

type#06 contains two instances of the same IP camera. It is worth noting that several devices connected to *IoT Gateway* through an intermediary gateway would be considered as a single device and would be allotted a single device type. Intermediary gateways are usually proprietary and connect devices from a same manufacturer that have also the same or similar purpose (e.g., light bulbs). We conclude that our technique to automatically assign device types is relevant since similar/same devices from the same manufacturers are likely to have same QoS and security requirements. These can be addressed with a same policy specific to our autonomously defined device types.

We demonstrate the accuracy of device-type identification using a 4-fold stratified cross-validation. We randomly split our 6,224 fingerprints into four equal subsets while respecting class (device type) distribution. We use three subsets for training our kNN identification model and test it on the remaining subset. This process is repeated four times to test each of the four subsets. We ran the cross-validation 10 times with random seeds. Figure 5 presents the precision, recall and f1-score for identifying each device type. All metrics reach over 0.95 for most devices. The overall accuracy of identification across all types is 0.982, showing its effectiveness. A confusion matrix presents detailed results for this experiment in Tab. V.

We computed the time required for identifying the type of a device. This process is divided into three stages. The first stage consists of capturing the traffic generated by the device, which lasts for a fixed duration of 30 minutes. The second stage consists of pre-processing and extracting the fingerprint from the traffic capture (steps 1+2 in Fig. 2), which lasts for 52.6 $ms \pm 36.5$ on average. The third stage is the classification of the fingerprint using kNN, which takes 0.1 *ms* on average. The duration of device identification is largely dominated by the time required for traffic capturing (30 minutes = 1,800,000 *ms*) that is 5 orders of magnitude longer than any of the other stages. The duration of traffic capture is static regardless of the number of devices to identify by *IoT Gateway* or the number of device types (classes) in the kNN model. Fingerprint extraction must be run for each device connected to an *IoT Gateway*. Let us assume that the *IoT Gateway* needs to be capable of identifying a few tens of IoT devices; running

TABLE V: Confusion matrix for device-type identification. Obtained with 10 repetitions of 4-fold cross validation. Columns represent predicted labels and rows actual labels.

	#01	#02	#03	#04	#05	#06	#07	#08	#09	#10	#11	#12	#13	#14	#15	#16	#17	#18	#19	#20	#21	#22	#23
type#01	480	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0
type#02	24	1240	0	10	3	5	0	0	16	103	0	0	0	3	50	0	26	0	0	0	0	0	0
type#03	0	0	3736	0	0	10	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0
type#04	0	0	0	2775	4	39	0	0	1	42	0	14	5	0	0	0	0	0	0	0	0	0	0
type#05	0	0	0	2	8707	12	0	0	10	19	0	0	0	0	0	0	0	0	0	0	0	0	0
type#06	0	0	0	10	0	3756	0	0	6	0	0	0	0	0	0	0	0	0	0	8	0	0	0
type#07	0	10	0	0	0	2	394	0	1	0	1	2	0	0	0	0	0	0	0	0	0	0	0
type#08	0	0	0	0	0	10	0	7390	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
type#09	0	0	0	0	0	0	0	0	2619	21	0	0	0	0	0	0	0	0	0	0	0	0	0
type#10	0	0	0	0	0	0	0	0	2	7323	0	0	0	17	0	0	0	0	0	0	0	0	8
type#11	0	0	0	13	38	30	0	0	0	1	2581	4	0	0	0	3	0	0	0	0	0	0	0
type#12	0	0	0	10	0	35	0	0	0	0	0	1955	0	0	0	0	0	0	0	0	0	0	0
type#13	0	0	0	0	0	0	0	0	0	21	0	0	3470	14	0	0	5	0	0	0	0	0	20
type#14	0	3	0	9	0	0	0	0	0	39	0	0	11	350	0	0	0	0	0	10	0	0	28
type#15	0	4	0	20	2	0	0	0	0	17	0	0	0	10	1412	0	0	0	0	0	0	0	5
type#16	0	0	0	2	0	3	0	0	0	0	0	18	0	0	0	2437	0	0	0	0	0	0	0
type#17	0	20	0	0	0	0	0	0	0	6	0	0	10	0	0	0	3884	0	0	0	0	0	0
type#18	0	0	0	50	0	0	0	0	12	0	1	0	0	0	0	0	12	1715	0	0	0	0	0
type#19	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	870	0	0	0	0
type#20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	870	0	0	0
type#21	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	1710	0	17
type#22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	2	0	292	0
type#23	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	10	0	1189

this process in parallel would take less than 1 second. The time for fingerprint classification using kNN increases linearly with the number of training samples in the kNN model. Assuming that the same number of instances is kept for every class in the model, the time for fingerprint classification would increase linearly with the number of classes (device types) in the kNN model. Our model containing 23 device types takes 0.1 *ms* to classify a fingerprint. Thus managing thousands of device types would take less than 1 second and device identification would still be largely dominated by the traffic capture which takes 30 minutes.

These experiments show that AUDI has high accuracy (98.2%) across all tested devices. All devices were identified within a fixed time of 30 minutes regardless of their operation mode and this time would remain the same even when considering a much larger number of device types to identify. AUDI meets the scalability requirement **R2** and the stability requirement **R3**.

C. Feature importance

We computed scores for feature importance to evaluate the impact of our 33 features on device-type identification. Since kNN does not provide information about features most useful in classification, we used the ReliefF feature selection algorithm [15] to compute these scores. While several methods exist for computing feature importance, e.g., information gain, PCA [18], we chose ReliefF because it is conceptually close to kNN, since its feature scoring is based on the differences in feature values between nearest neighbor instance pairs.

Table I presents the importance score for each feature. All four *period duration* features have high scores, which shows that IoT devices of different types have periodic flows with very different durations. The counts of periodic flows (*f1*-*f2*) are also highly relevant features meaning that IoT devices of different types have different numbers of periodic flows. The most relevant feature is the count of flows with a static source port (*f7*). This means that IoT devices are heterogeneous in the way they manage their periodic communications: some keep an open connection over time while others periodically re-initiate a new connection for the same flow. While some features have a low importance (e.g., *f3-f4-f11-f12*), they slightly improve the accuracy of device-type identification and we decided to keep them in our set of features. A large set of features (and some feature redundancy) also increases the resilience of machine learning based systems to adversarial machine learning attacks such as data poisoning [19].

D. Learning time

To show that our identification model can be quickly learned, we evaluate its accuracy with a varying amount of training data. As presented in Sect. IV-B, we selected $k = 5$ as minimum number of components for a class in kNN. Figure 6 depicts the increase in precision, recall and f1-score as we vary the size of the training set from 5 fingerprints per device (2.5 hours monitoring) to 40 fingerprints per device (20 hours monitoring). We see that the accuracy in all metrics increases quickly from 0.87 to 0.95 but then stabilizes with a small gradient. It shows that after a few ($\approx$ 12) hours of monitoring, more training data does not significantly increase accuracy. This time is likely even shorter considering that several *IoT Gateway*s contribute training data (fingerprints) for each device type in parallel. This shows that learning an effective device identification model requires only a few hours of traffic monitoring *globally*.

To summarize, we showed that our method for automatically learning device type is relevant. We demonstrate that the identification technique is effective and accurate, even when using little training data, which makes it fast at identifying

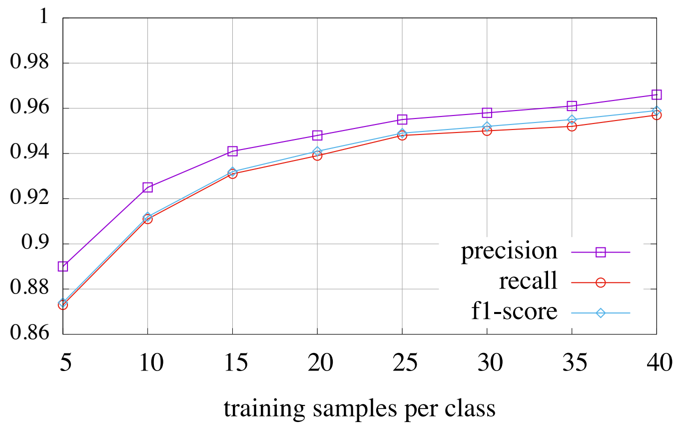

Fig. 6: Precision, recall and f1-score increase with respect to training set size.

newly released IoT devices. AUDI meets a second aspect of requirement **R3**: learning model for device types quickly.

VI. Security Analysis and Discussion

Spoofing device fingerprints. A compromised device can attempt to modify its background traffic such that its fingerprint changes and it gets identified as another device type. Fingerprinting can be implemented as a one-time operation performed when a new IoT device is detected in the network. It is reasonable to assume that brand new IoT devices are not compromised when they are first added to a SOHO network. AUDI requires only 30 minutes to accurately identify the type of a device. Spoofing of a targeted device fingerprint requires the attacker to generate new periodic communication and to disable existing periodic communication. The latter impacts the functionality of a device, which may be detected as compromised by its user (e.g., missing periodic report) or its cloud service provider (e.g., missing reception of periodic heartbeat signal). In addition, spoofing a fingerprint essentially results in the device being assigned a different policy than the one it was supposed to get. Consequently, the device will nevertheless be constrained by a policy enforced by *IoT Gateway*. For example, when device identification is used for anomaly detection [12], a device spoofing its fingerprint is likely to be flagged by the anomaly detection component of *IoT Gateway* since the communication profile of the device (legitimate or otherwise) is unlikely to be included in the anomaly detection profile of the (incorrect) device type.

Spoofing MAC address. Device identification is based on monitoring layer-2 traffic involving a particular device, identified by its MAC address. An adversary who has compromised a device can attempt to evade identification by spoofing its MAC address in the packets it sends out. MAC address spoofing can be mitigated using additional techniques for fingerprinting hardware interfaces on wireless [8], [11] and on wired connections [20]. These build a unique signature for the packets sent by a device related to hardware characteristics. Such fingerprints are difficult to spoof [21]. Alternatively, secured association protocols like WiFi Protected Setup provided by WPA2 [22] can be used to associate IoT devices to *IoT Gateway*. Such association protocols require user involvement (e.g., physically pushing a button on the gateway) to associate a new device to the access point. The association results in a device-specific shared key that can be subsequently used by the gateway to authenticate the device. This prevents rogue devices from connecting to the network by spoofing the MAC address of a device already associated with *IoT Gateway*. We conclude that AUDI meets the security requirement **R4**.

Generalizability of device fingerprinting. Features that compose our device fingerprint have been defined to model periodic flows and to differentiate IoT devices having different periodic flows. This feature definition and the use of a specific classifier, kNN, was motivated in Sect. IV. As in any machine learning application, the efficacy of a feature set and a classifier can only be demonstrated for a specific task and a specific dataset (no free lunch theorem [23]).

To ensure generalizability, we defined fingerprint features and selected a kNN classifier without prior knowledge about communications of specific IoT devices. Consequently our features are independent from any dataset and more specifically from the data we later processed in experiments. Data-independent features and the machine learning method choice ensure generalizability of the fingerprinting technique [24]. Having assessed our technique on a large set of 33 IoT devices (IP cameras, sensors, coffee machine, etc.) representative of typical smart home IoT devices, we expect that the high efficacy (98.2% accuracy) seen during our evaluation (cf. Sect. V-B) is likely to be generalizable to other IoT devices.

Some IoT devices, especially those that operate on battery power, may be kept turned off by default and activated only on explicit user triggers. Such devices naturally will not have periodic communications; consequently techniques like AUDI are not effective in identifying such devices. We had two such devices (out of 35) in our lab: two smart scales that we discarded for experiments. These devices were normally powered off and generated communication traffic only when activated by a physical user interaction. No other action, e.g., incoming communication from the cloud service, companion app, or other local devices, could activate them otherwise. Physical user interactions with these devices are typically infrequent, which explains why they are designed to be battery powered. Consequently, these devices are not critical from security or network management perspectives. They only generate low volume of communication on infrequent occasions. Also, they are unlikely to be discovered by IoT malware scanning the local network since they are turned off most of the time.

VII. Related Work

Early work in wireless communication fingerprinting targeted the identification of hardware- and driver-specific characteristics [9], [10], [25]. IoT-oriented device identification techniques leverage sensor-specific features [26]–[29] to uniquely identify a device. Our identification technique is positioned between the former and latter approaches, providing the right granularity to passively identify *device types*.

Some solutions address device-type identification with the same granularity as we do [7], [30]–[33], while considering different definitions of "type". GTID [7] identifies the make and model of a device by analyzing the inter-arrival time of packets sent for a targeted type of traffic (e.g. Skype, ICMP, etc.). GTID requires a lot of traffic over several hours to identify a device's type. Aksu et al. [32] also model the inter-arrival time of Bluetooth packets to identify different model of wearable devices from a smartphone. Maiti *et al.* [6] introduced a device-type identification technique relying on analysis of encrypted WiFi traffic. A Random Forest classifier is trained with features extracted from a long sequence of WiFi frames. The technique was evaluated on 10 IoT devices and required at least 30,000 frames to be effective. In standby mode an IoT device can take days to generate such volumes of traffic. IoT Sentinel [5], [34] leverages the burst of network traffic typical for the setup phase of an IoT device to identify its type. While accurate and requiring only two minutes of monitoring, IoT Sentinel only operates when a device is first installed to a network. Meidan *et al.* [31] analyze TCP sessions to identify generic types of IoT devices, i.e., *smoke sensor*, *baby monitor*, etc. The observation of at least 20 TCP sessions was required to reach acceptable accuracy for 17 devices. The authors reported that 1/3 of their IoT devices did not produce any TCP sessions without user interactions (i.e., in standby mode), and for the remaining 2/3 the mean inter-arrival time of TCP sessions was up to 5 minutes, requiring over one hour and a half to be identified. Guo and Heidemann [33] use our same intuition to identify IoT devices, namely that IoT devices periodically connect to specific services on the Internet. They identify the server names and IP addresses that a known IoT device connects to on the Internet. This information is later used to identify unknown devices if they connect to the same IP addresses. A limitation of this approach is that different IoT devices from a same manufacturer often connect to the same servers which produces collisions between device types from a same manufacturer. Also many IoT device manufacturers leverage cloud services such as Amazon for hosting their services [30], which can also produce collisions.

State-of-the-Art methods for device-type identification are supervised and require labeled data to be trained. AUDI is not restricted to a finite set of pre-learned device types. It creates abstract device types, learns their fingerprints and adapts autonomously when new types are discovered. AUDI is also not restricted to a specific type of dense network traffic. It is the first technique to identify IoT device types based on their background periodic communication. Consequently and in contrast to previous work, it identifies the type of an IoT device under *any state of operation*.

Some security solutions for the IoT with a distributed design close to AUDI have been proposed in commercial solutions, e.g., IoT guardian from Zingbox [35]. While relying on an unsupervised device identification technique, IoT guardian does not propose any concrete implementation for it. Moreover, IoT guardian relies on partial deep packet inspection, which prevents it from being used on encrypted communications. AUDI does not have such limitations.

VIII. Conclusion

Identification of devices that compose a network, or network mapping, is the basis for many network management applications ranging from network resource allocation and network slicing to security management. In this paper we introduced AUDI, a novel autonomous approach for identifying the type of devices in IoT networks. AUDI generates abstract device-type labels to be used as input for such self-learning systems working without human supervision. We have built an autonomous anomaly detection system based on AUDI (described in the longer research report [12]). We hope that AUDI can pave the way for other novel autonomous approaches for managing IoT networks.

A future improvement to increase the autonomy of AUDI is to make it self-parametrizable. The four clustering parameters of AUDI can be self-defined and optimized in real-time by the system rather than using, e.g., a training period in a lab setup. This would remove any need for user involvement, even prior to system deployment.

We introduced fingerprints for IoT device types that are derived from their periodic communications. Future work can focus on defining fingerprints that are specific and tied to the envisioned network management application for device identification. Network management and security policies could be automatically derived from such device fingerprints, e.g., as a function of the fingerprint, rather than using our proposed linkage between fingerprint and policy.

Acknowledgments: This work was supported in part by the Academy of Finland (SELIoT project - grant 309994), the German Research Foundation (DFG) within CRC 1119 CROSSING (S2 and P3) and the Intel Collaborative Institute for Collaborative Autonomous and Resilient Systems (ICRI-CARS). We would also like to thank Cisco Systems, Inc. for their support of this work.

References

[1] M. Antonakakis, T. April, and et al., "Understanding the mirai botnet," in *26th USENIX Security Symposium*, 2017, pp. 1093–1110.
[2] S. Edwards and I. Profetis, "Hajime: Analysis of a decentralized internet worm for IoT devices," Rapidity Networks, Tech. Rep., 2016.
[3] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.
[4] T. Sauvage, "Authenticated root os command execution," 2016.
[5] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma, "IoT Sentinel: Automated Device-Type Identification for Security Enforcement in IoT," in *37th IEEE International Conference on Distributed Computing Systems*, 2017.
[6] R. R. Maiti, S. Siby, R. Sridharan, and N. O. Tippenhauer, "Link-layer device type classification on encrypted wireless traffic with cots radios," in *European Symposium on Research in Computer Security*, 2017, pp. 247–264.
[7] S. V. Radhakrishnan, A. S. Uluagac, and R. Beyah, "GTID: A technique for physical device and device type fingerprinting," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 5, pp. 519–532, 2015.
[8] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 93–108, April 2005.
[9] J. Cache, "Fingerprinting 802.11 implementations via statistical analysis of the duration field," *Uninformed*, vol. 5, 2006.
[10] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. Van Randwyk, and D. Sicker, "Passive data link layer 802.11 wireless device driver fingerprinting," in *USENIX Security Symposium*, 2006.
[11] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *ACM International Conference on Mobile Computing and Networking*, 2008, pp. 116–127.

[12] T. D. Nguyen, S. Marchal, M. Miettinen, N. Asokan, and A. Sadeghi, “DÏoT: A self-learning system for detecting compromised IoT devices,” *CoRR*, vol. abs/1804.07474, 2018.

[13] I. Hafeez, A. Y. Ding, and S. Tarkoma, “Securing edge networks with securebox,” *arXiv preprint arXiv:1712.07740*, 2017.

[14] S. Winograd, “On computing the discrete fourier transform,” *Mathematics of computation*, vol. 32, no. 141, pp. 175–199, 1978.

[15] I. Kononenko, E. Šimec, and M. Robnik-Šikonja, “Overcoming the myopia of inductive learning algorithms with ReliefF,” *Applied Intelligence*, vol. 7, no. 1, pp. 39–55, 1997.

[16] R. J. Samworth *et al.*, “Optimal weighted nearest neighbour classifiers,” *The Annals of Statistics*, vol. 40, no. 5, pp. 2733–2763, 2012.

[17] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[18] H. Abdi and L. J. Williams, “Principal component analysis,” *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[19] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, “Is feature selection secure against training data poisoning?” in *International Conference on Machine Learning*, 2015, pp. 1689–1698.

[20] R. M. Gerdes, T. E. Daniels, M. Mina, and S. Russell, “Device identification via analog signal fingerprinting: A matched filter approach.” in *NDSS*, 2006.

[21] C. Arackaparambil, S. Bratus, A. Shubina, and D. Kotz, “On the reliability of wireless fingerprinting using clock skews,” in *ACM Conference on Wireless Network Security*. ACM, 2010, pp. 169–174.

[22] WiFi Alliance, *WiFi Simple Configuration Technical Specification*.

[23] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *IEEE Trans. on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[24] S. Marchal, G. Armano, T. Gröndahl, K. Saari, N. Singh, and N. Asokan, “Off-the-hook: An efficient and usable client-side phishing prevention application,” *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1717–1733, 2017.

[25] C. Maurice, S. Onno, C. Neumann, O. Heen, and A. Francillon, “Improving 802.11 fingerprinting of similar devices by cooperative fingerprinting,” in *International Conference on Security and Cryptography*, 2013, pp. 1–8.

[26] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, “Mobile device identification via sensor fingerprinting,” *arXiv preprint:1408.1416*, 2014.

[27] T. Van Goethem, W. Scheepers, D. Preuveneers, and W. Joosen, “Accelerometer-based device fingerprinting for multi-factor mobile authentication,” in *International Symposium on Engineering Secure Software and Systems*, 2016, pp. 106–121.

[28] Y. Sharaf-Dabbagh and W. Saad, “On the authentication of devices in the Internet of Things,” in *IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks*, 2016, pp. 1–3.

[29] I. Haider, M. Höberl, and B. Rinner, “Trusted sensors for participatory sensing and IoT applications based on physically unclonable functions,” in *ACM International Workshop on IoT Privacy, Trust, and Security*, 2016, pp. 14–21.

[30] N. Apthorpe, D. Reisman, and N. Feamster, “A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic,” *arXiv preprint arXiv:1705.06805*, 2017.

[31] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, “Detection of unauthorized IoT devices using machine learning techniques,” *CoRR*, vol. abs/1709.04647, 2017.

[32] H. Aksu, A. S. Uluagac, and E. Bentley, “Identification of wearable devices with bluetooth,” *IEEE Trans. on Sustainable Comput.*, 2018.

[33] H. Guo and J. Heidemann, “Ip-based iot device detection,” in *Workshop on IoT Security and Privacy*, 2018, pp. 36–42.

[34] M. Miettinen, S. Marchal, I. Hafeez, T. Frassetto, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, “IoT Sentinel demo: Automated device-type identification for security enforcement in iot,” in *IEEE International Conference Distributed Computing Systems*, 2017, pp. 2511–2514.

[35] G. Cheng, P.-C. Yip, Z. Xiao, R. Xia, and M. Wang, “Packet analysis based IoT management,” 2016, uS Patent App. 15/087,861.

Samuel Marchal is a post-doctoral researcher in the secure systems research group at Aalto University. He received the M.Sc. degree in computer science in 2011 from TELECOM Nancy, France. He received the Ph.D. degree in 2015 jointly from the University of Luxembourg and the University of Lorraine, France. He conducted his doctoral research at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) in Luxembourg. His interests lie in system security, network security and machine learning.

Markus Miettinen is a postdoctoral researcher at the System Security Lab of Technische Universität Darmstadt, Germany. He received his M.Sc. from the University of Helsinki, Finland in 2002 and graduated as Dr.-Ing. from TU Darmstadt in 2018. Before joining academia, he acquired more than a decade of professional experience in industrial research at the Nokia Research Center in Helsinki, Finland and Lausanne, Switzerland. His research interests lie in utilizing machine learning methods for realizing autonomous security solutions for IoT and mobile computing environments..

Thien Duc Nguyen is a research assistant and a PhD candidate at the System Security Lab of Technische Universitat Darmstadt, Germany. He is currently working in various topics related to machine learning-based security mechanisms for IoT, contextual security and mobile security.

N. Asokan is a professor of computer science at Aalto University where he co-leads the secure systems research group and directs Helsinki-Aalto Center for Information Security. Before joining academia, he spent 17 years in industry research labs with IBM Research and Nokia Research Center. Asokan received his formal education from University of Waterloo, Syracuse University, and Indian Institute of Technology, Kharagpur. https://asokan.org/asokan/

Ahmad-Reza Sadeghi is a full professor for Computer Science at Technische Universität Darmstadt, where he directs the System Security Lab and the Intel Collaborative Research Institute for Autonomous and Resilient Systems. He received his PhD from the University of Saarland. He has served on the editorial board of ACM TISSEC and as Editor-in-Chief for IEEE Security and Privacy Magazine. https://www.trust.informatik.tu-darmstadt.de/people/ahmad-reza-sadeghi/