DEFT: A Distributed IoT Fingerprinting Technique

Vijayanand Thangavelu, Dinil Mon Divakaran, *Senior Member, IEEE*, Rishi Sairam, Suman Sankar Bhunia, and Mohan Gurusamy, *Senior Member, IEEE*

***Abstract*—Identifying IoT devices connected to a network has multiple security benefits, such as deployment of behavior-based anomaly detectors, automated vulnerability patching of specific device types, dynamic attack mitigation, etc. In this paper, we look into the problem of IoT device identification at network level, in particular from an ISP's perspective. The simple solution of deploying a supervised machine learning algorithm at a centralized location in the network neither scales well nor can identify new devices. To tackle these challenges, we propose and develop a distributed device fingerprinting technique (DEFT), a distributed fingerprinting solution that addresses and exploits the presence of common devices, including new devices, across smart homes and enterprises in a network. A DEFT controller develops and maintains classifiers for fingerprinting, while gateways located closer to the IoT devices at homes perform device classification. Importantly, the controller and gateways coordinate to identify new devices in the network. DEFT is designed to be scalable and dynamic—it can be deployed, orchestrated, and controlled using software-defined networking and network function virtualization. DEFT is able to identify new device types automatically, while achieving high accuracy and low false positive rate. We demonstrate the effectiveness of DEFT by experimenting on data obtained from real-world IoT devices.**

***Index Terms*—Fingerprint, identification, IoT, network, network function virtualization (NFV), security.**

I. Introduction

SECURITY concerns with respect to IoT devices are different and new, in comparison with the PC market. A primary factor is the scale—the number of IoT devices is expected to grow to a few tens of billions by 2023 [1]. Besides, the IoT market is much more heterogeneous in the types, based (mostly) on applications and protocols. The number of vendors in this market is continuously increasing, but not all vendors would continue to exist during the lifetime of their products. These factors raise multiple issues. As new vulnerabilities get discovered, it is likely that many of these will be left unpatched, and hence open to exploits. Even in the PC market, where there is often a user to maintain the computer(s) owned, it is not uncommon that many vulnerabilities remain unpatched. This trend will be worse for IoT devices, as most devices will not have a human user dedicatedly attending to it. Exposed vulnerabilities are exploited by attackers for a variety of purposes, such as leaking private information [2], launching large-scale attacks (as witnessed in the recent Mirai attack [3]), infecting critical infrastructures [4], etc. Therefore, to secure IoT devices, multiple approaches need to be taken, from design to attack monitoring to mitigation.

In this paper, we take a network-centric approach for securing IoT devices. The first step in this process is to identify IoT devices. This is also referred to as *fingerprinting* of devices, as solution seeks to find unique characteristics that can identify and distinguish devices. Identifying IoT devices has multiple security benefits.

1) Knowing the devices in a network helps in analyzing the normal behavior of a particular device type, which can subsequently be used to detect anomalous behaviors, such as unusual incoming traffic, connections to unseen servers, etc.
2) Once a vulnerability of a device type (say, camera of vendor *X* and model *Y*) is known, security patches or mitigation solutions can be applied at the earliest, specifically on the devices identified to be of this type.

In addition, device identification helps an organization to continuously maintain its asset list, to quarantine and isolate misbehaving or vulnerable devices, etc. If fingerprints of devices are unique, then they can also be used for authentication [5].

We study the problem of fingerprinting IoT devices by analyzing network traffic. We view the problem from the perspective of an ISP; specifically, we focus on the scenario where IoT devices are connected to an ISP network via *gateways*. Here, gateways are essentially access points (APs) at smart homes and enterprises, connecting IoT devices to the Internet. In the near future, a gateway is likely to talk multiple communications protocols, such as Ethernet, Wi-Fi, Bluetooth, ZigBee, etc., so that it can communicate with a variety of IoT devices.

A centralized approach for solving the problem of fingerprinting IoT devices is to train a supervised machine learning model using network traffic of devices, and use that model at a centralized controller in the network, to classify devices based on the traffic characteristics. We note that, a good solution needs to update its model continuously, as the traffic patterns of devices change over time due to reasons such as change in configuration, firmware update, user behavioral changes, environmental changes, etc. This means that, without any other intelligence in the network, a centralized fingerprinting

Manuscript received May 28, 2018; revised July 8, 2018; accepted August 7, 2018. Date of publication August 15, 2018; date of current version February 25, 2019. This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd. *(Corresponding author: Dinil Mon Divakaran.)*

V. Thangavelu, R. Sairam, S. S. Bhunia, and M. Gurusamy are with the Electrical and Computer Engineering Department, National University of Singapore, Singapore (e-mail: vijayanand@u.nus.edu; rishi.sairam@u.nus.edu; suman.s.bhunia@ieee.org; gmohan@nus.edu.sg).

D. M. Divakaran is with the Cyber Security R&D Division, Singtel, Singapore (e-mail: dinil.divakaran@singtel.com).

Digital Object Identifier 10.1109/JIOT.2018.2865604

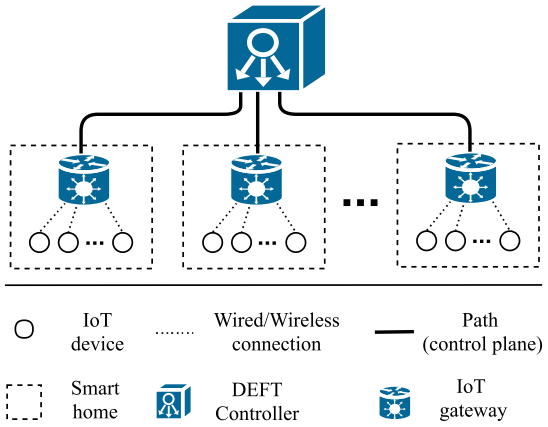

Fig. 1. Hierarchical IoT network architecture.

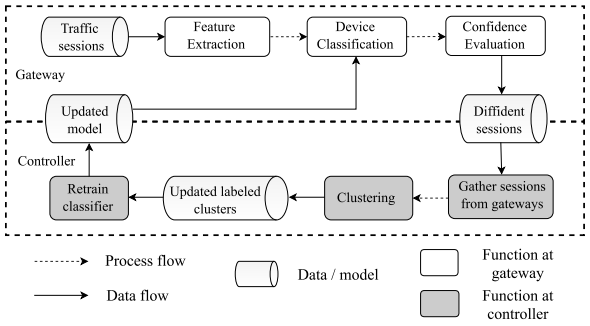

Fig. 2. Block diagram showing DEFT components.

solution has a serious limitation—all traffic flows from devices need to be sent to the controller. This is challenging to scale with increasing number of devices and smart homes. Besides, a supervised classifier only identifies known devices (using pretrained models). However, in an ever growing IoT market, it is important to detect new devices as well.

We argue that a fingerprinting solution for IoT devices should meet two important objectives: 1) the fingerprinting solution should be scalable while not compromising on accuracy of device identification and 2) the solution should be able to dynamically detect and differentiate new IoT devices connected to the network. The solution should learn the fingerprints of new devices automatically.

In this paper, we propose and develop a distributed device fingerprinting technique (DEFT) for identifying IoT devices. DEFT works on a hierarchical network architecture, consisting of two entities: 1) control logic and 2) gateway. A DEFT control logic resides in the ISP network, and controls a set of gateways for fingerprinting IoT devices. Fig. 1 illustrates this architecture. Each gateway corresponds to a smart home or enterprise to which IoT devices are connected. As ISPs are gradually embracing software-defined networking (SDN), the DEFT control logic is essentially an application deployed at an SDN controller. The path shown in the figure is therefore a path in the control plane of the ISP network, connecting the SDN-controller to the gateway. Henceforth, unless otherwise stated, we use *controller* to refer to the DEFT control logic. Though one can imagine a tree structure of controllers, where a controller higher up in the tree structure controls all its child nodes, we limit our discussion henceforth to two levels—the first level consists of (independent) controllers and the second level consists of nonoverlapping sets of gateways, each set controlled by a DEFT controller.

DEFT is designed to meet the two objectives stated above. The gateways perform device classification using the model developed by the controller. All traffic sessions from IoT devices are processed and classified locally at the corresponding gateway, without being sent to the controller. The only information sent from a gateway to its controller is the feature vector corresponding to *selected* traffic sessions. This happens rarely, when the gateway is not confident about its classification (and most likely when a new device is connected to the gateway). Besides, the size of a feature vector is limited to a few KBs, whereas that of a traffic session can vary arbitrarily from hundreds of KBs to hundreds of MBs and even more.

The DEFT controller learns fingerprints of devices, builds and continuously improves the fingerprint classification system, to deploy the up-to-date classifier at the gateways it controls. The controller coordinates with the gateways to gather information about (potentially) new devices across the different smart homes/enterprises, to subsequently label and identify new devices. DEFT is designed to work with SDN and network function virtualization (NFV), with the controller having the ability to orchestrate the classifier as a virtual network function (VNF) at the gateways and reinforce the classifier model as and when required. In our design, the DEFT controller can also be deployed at a private cloud of the ISP, as the controller does not process network traffic. We evaluate and demonstrate the effectiveness of DEFT in fingerprinting real-world IoT devices with high accuracy. During the process, we also analyze different traffic features useful in fingerprinting devices.

The rest of this paper is organized as follows. We present an overview of our proposed system DEFT in the next section. We provide details on the features we use for identifying IoT devices using network traffic in Section III. In Section IV, we develop and present the algorithms that define DEFT. We evaluate the effectiveness of DEFT in fingerprinting devices and present the results in Section V.

II. Overview of DEFT

Fig. 2 gives an overview of DEFT building blocks. The upper part of the figure consists of processes executed at the gateways, while the processes in the lower part are executed at the DEFT controller. Features are extracted from traffic sessions (defined later in Section III) that arrive at the gateway. After feature extraction, the gateway classifies the device (traffic session) into a particular type (or class). The classifiers used at the gateways are obtained from the controller. The gateways do not perform any training; instead training of classifiers is performed by the controller. After performing classification, the gateway separates out the sessions that were classified with low probability, as *diffident sessions*; such sessions are later sent to the controller. A classification may have low probability when the corresponding device is new; or the probability

could be low even for an existing device when the traffic pattern has changed. Note that, a change in the firmware of an existing device might lead to a low classification probability.

All the three modules of *feature extraction*, *device classification*, and *confidence evaluation* are parts of a single fingerprint classification application that can be developed as a VNF and deployed at the gateway. Furthermore, three individual modules can be developed as microservices, thereby allowing them to be modified and deployed independently, while adhering to standard interfaces for communications between themselves. Thus, our design is based on NFV and microservice architecture to provide modularity and robustness.

The controller gathers *diffident sessions* from the different gateways it controls, and proceeds to cluster them. For clustering, the controller uses a set of labeled data as *seed* clusters. The controller updates the labeled dataset continuously and actively based on its algorithm; and each time the dataset is updated, the controller retrains the classifier generating a new model. This model is sent to all the gateways it controls. And the gateways use the updated model for classifying the traffic sessions captured from then on.

As we will discuss later, a gateway only sends *feature vectors* of diffident sessions to its controller, and not the actual traffic session. The size of a feature vector is $\mathcal{O}(k)$, where k is the dimension of the vector. In this paper, the value of k is around 100. Thus, even when sessions may have hundreds of packets resulting in session sizes that varies between hundreds of KBs (kilobytes) and MBs, the size of the corresponding feature vector is in 1–2 KBs.

It is also worth noting that, DEFT is designed to be scalable. If the number of gateways increase, more instances of the DEFT controller can be spawned to balance the load; and the different controllers can synchronize the classifier model via a controller in the above layer in the hierarchical architecture.

A. Illustration of the Fingerprinting Process

We illustrate the fingerprinting process in DEFT using a simple example depicted in Fig. 3. There are three gateways—G_1, G_2, and G_3—connected to a DEFT controller. The system currently has a model for classifying six devices {a, b, c, d, e, f}. The devices currently connected to the gateways are shown in the figure. The next steps, in simplified form, are given below.

1) A new device, of type x, connects to the gateway G_1.
2) G_1 observes that the classification probability of traffic session from this device is low, based on the current model in use.
3) G_1 collects the traffic sessions (feature vectors) of this device and sends to the controller.
4) Meanwhile, a new device, of the same type x, is connected to the gateway G_3.
5) G_3 has low confidence in classifying this new device's sessions.
6) G_3 sends traffic sessions of this device to the controller.
7) The controller, on receiving sufficient number of sessions (data points), performs clustering, unaware that the sessions obtained from the two gateways (G_1 and G_3) could be of the same device type. The clustering process, however groups the data points from G_1 and G_2 into a new and distinct cluster. It updates the labeled dataset and retrains the model based on the clustering output.
8) Subsequently, the controller sends the new model to all the gateways.
9) A new device, of type x, connects to the gateway G_2.
10) G_2 classifies x using the new model (to the label learned by the model which the controller trained).

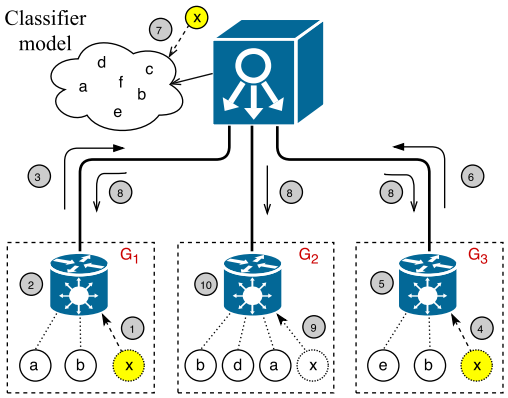

Fig. 3. Illustration of the fingerprinting system.

III. Traffic Features for IoT Fingerprinting

Network traffic has important information that can be extracted as *features* for different kinds of analysis, including fingerprinting. Features could be extracted both from packet headers as well as payloads. Extracting features from payloads is an expensive process, though such features may provide useful information for fingerprinting. Payload analysis is also often considered intrusive to user privacy. The features used in DEFT are not from user data; however, we do extract features from DNS queries and HTTP URI's (see Section III-A), which may be considered as private information.

Depending on the analysis, traffic can be represented at different levels. At the finest level of granularity, a feature vector represents per packet information; examples being application protocol (port number) used, packet size, destination IP address, IP options, etc. This representation is useful if the goal is to classify each packet. However, there is no strong motivation to carry out fingerprinting of devices at packet level, in particular considering the fact that per-packet classification is costly. Instead, we find it sufficient to classify a *traffic session*. A session is an aggregation of traffic connections from and to one particular device and localized in time. For example, if the session time is defined to be 15 min, then all the network packets sent and received by a device within a 15-min window will constitute a single session. Therefore, a session is identified by both a device and a time interval.

Session interval can be determined either dynamically or statically. In a dynamic approach, one can look for an inactive period, and group traffic connections into one session as long as the inactivity period is less than a predetermined time (say, 1 min). In a static approach, time is split into fixed-size intervals (say, 15 min), and all traffic connections made by a

device within one such interval is considered as a single session. The length of dynamic sessions could be quite arbitrary, particularly because devices may maintain keep-alive probes. For this paper, we define sessions based on fixed-size time intervals.

A. Feature List

The features used in DEFT are listed in Table I. We broadly classify them below. In the following as well as in the table, we use "stats" and "statistics" to denote the mean, minimum, and maximum of the concerned feature in a session.

1) *V_1:* DNS being one of the fundamental protocols of the Internet, features extracted from DNS protocol could be valuable for fingerprinting IoT devices. We consider features related to both DNS and multicast DNS. The features considered are number of DNS queries, DNS packet count, most frequently queried domain name, number of DNS errors (i.e., response code $\neq$ NOERROR), the number of INTERNET class queries, statistics of DNS packet length and DNS query response time, etc. All these features are collected over each of the sessions defined based on time.
2) *V_2:* We also extract two session level features that are protocol agnostic: a) number of packets sent during the session and b) activity period of the session. Although a session might be defined for, say, 15 min, packets might be sent only in the first 10 min; in this case, the activity period is 10 min. Observe, this set of features is least privacy-intrusive, as they do not require reading of even packet headers.
3) *V_3:* Most communications between IoT devices and cloud are encrypted. As we see increasing adoption of TLS [6], we extract TLS related features for fingerprinting. The features extracted are minimum, maximum, and mean of TLS packet length, flow duration, and number of TCP keep-alive probes used in TLS session. In this class we also consider HTTP features as listed in the table.
4) *V_4:* In this feature class, we consider a number of protocols, as given in Table I. Session traversal of UDP through NAT (STUN) is used to establish bidirectional communication between an IoT device and its cloud server, in the presence of an NAT server. Simple service discovery protocol (SSDP) is a server-less discovery protocol, forming the basis of universal plug and play architecture; it is adopted by many IoT devices. Message queue telemetry transport (MQTT) is a publish–subscribe-based light-weight messaging protocol used to collect and transfer data from devices to their servers. Due to its various features (small footprint, adaptability with constrained network, simplicity in implementation, etc.), MQTT is expected to be widely adopted by the IoT market. Quick UDP Internet connection is a recently developed transport protocol, supported by Google servers and Google Chrome, that aims to perform better than the widely used TCP. We refer readers to Table I for the full list of features.

TABLE I
LIST OF TRAFFIC FEATURES EXTRACTED

ID	Protocols	Features
V_1	DNS mDNS	- # DNS queries - # DNS errors - # IN class queries - DNS packet count - Mode of domain name queries - DNS packet length (stats) - DNS query response time (stats)
V_2	Session statistics	- # packets in session - active duration
V_3	TLS	- # TCP keep alive - TLS packet count - TLS packet length (stats) - TLS flow duration (stats)
	HTTP	- # HTTP response code - # HTTP method - # TCP keep alive - HTTP request URI - HTTP packet count - HTTP packet length (stats) - HTTP flow duration (stats)
V_4	SSDP	- SSDP packet count - SSDP packet length (stats) - SSDP flow duration (stats)
	QUIC	- QUIC packet count - QUIC packet length (stats) - QUIC flow duration (stats) - # QUIC packet type
	MQTT	- MQTT packet count - MQTT packet length (stats) - MQTT flow duration (stats) - # MQTT packet type
	STUN	- STUN packet count - STUN packet length (stats) - STUN flow duration (stats)
	NTP	- NTP packet count - NTP packet length (stats) - NTP flow duration (stats)
	BOOTP	- BOOTP packet count - BOOTP packet length (stats) - BOOTP flow duration (stats)

All the above features form our feature vector of dimension 111. While previous works have used combinations of some of the feature mentioned above, to the best of our knowledge, all the features have not been used together for fingerprinting previously.

B. Initial Feature Analysis

To understand the goodness of the features in distinguishing the IoT devices, we carry out preliminary analysis using principal component analysis (PCA). PCA is often used for dimension-reduction, by explaining the variance in the data using a small number of orthogonal components. It is also useful to visualize high-dimension data in smaller, two or three, dimensions. We extracted features from traffic of 16 devices (listed in Table III) and analyzed them using PCA with two components; the corresponding 2-D planes are plotted in Fig. 4. For clarity, the devices are separated into smaller groups and presented in different plots. We observe that there is clear separation among feature vectors belonging to different

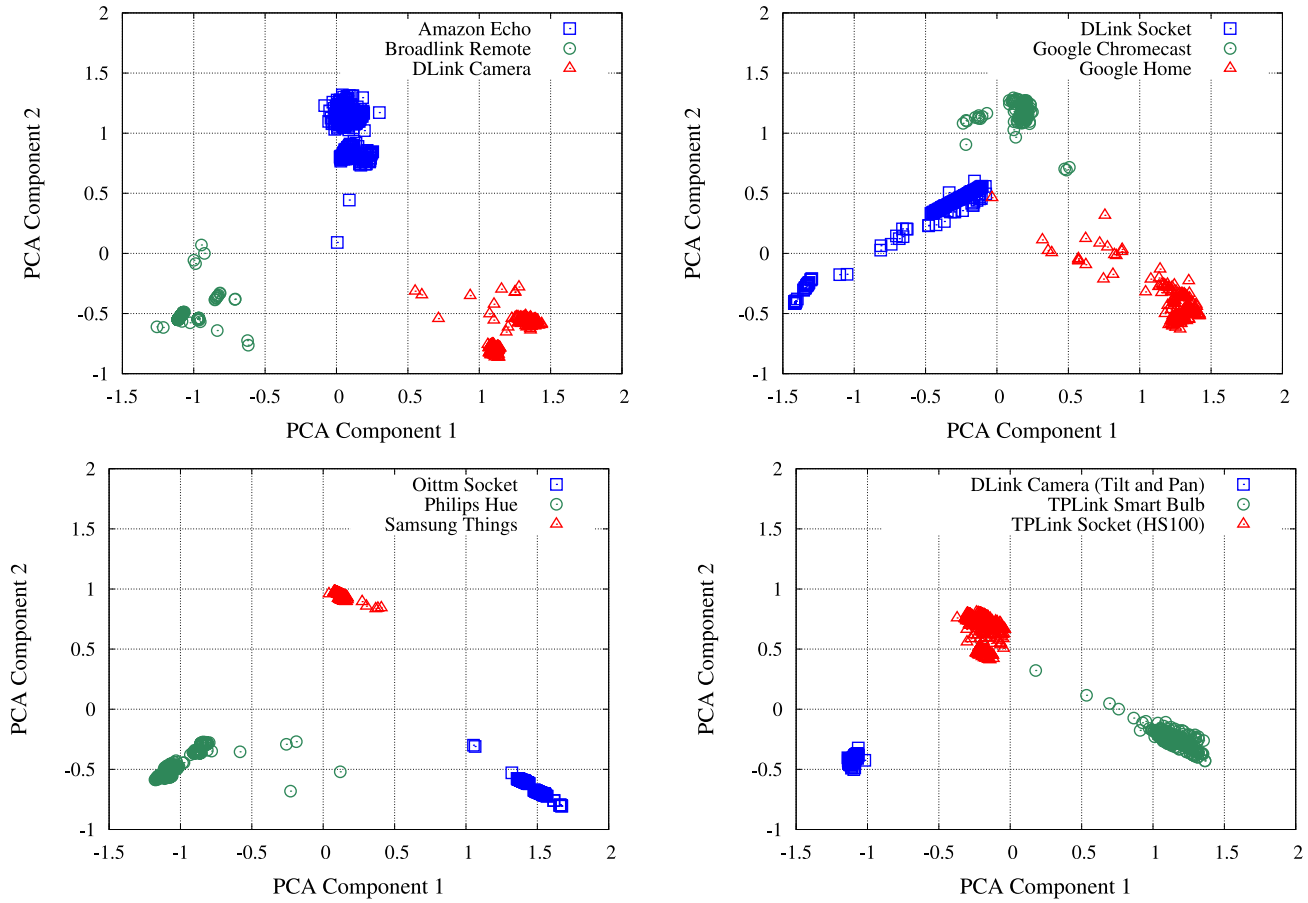

Fig. 4. PCA for different devices.

devices. Though we observed overlap when we plotted the components of all devices in a single 2-D plane, the overlap reduced significantly when another component was added to the PCA. This insight provides motivation to develop clustering algorithm(s) at the controller for identifying new devices. We use PCA only for this preliminary analysis, and do not use it further in any of the algorithms forming DEFT.

IV. DEFT: Algorithms

We develop the algorithms defining the DEFT system in this section. Before doing so, we provide background on related concepts. Table II gives the list of commonly used notations in this paper.

A. Preliminary

We brief some important concepts used in this paper, primarily to estimate the distance between two sets of points, so as to decide whether they are close enough to be considered as one cluster or not.

1) Centroid: Formally, given a set $\mathcal{F}$ of m-dimensional vectors, where $|\mathcal{F}|$ denotes the number of vectors in $\mathcal{F}$, the centroid ζ of set $\mathcal{F}$ is computed as

$$\zeta(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{\mathbf{f} \in \mathcal{F}} \mathbf{f} \tag{1}$$

the addition and average being computed over the vector components.

TABLE II
Table of Notations

Notation	Description
$\mathbf{F}$	Feature vector
m	Feature dimension
k	No. of clusters
$\mathcal{F}$	Aggregate of feature vectors
$\mathcal{M}$	Model for supervised classification
$\mathbb{P}$	List of feature aggregates
$\mathbb{T}$	List of merged feature aggregates
$\mathcal{R}$	List of clusters

2) Euclidean Distance: We also utilize the commonly used Euclidean distance, to compute the distance between two sets of points. Given two vectors $\mathbf{u}$ and $\mathbf{v}$ with the same dimension m, the Euclidean distance is computed as

$$\texttt{dist}(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^{m} (u_i - v_i)^2} \tag{2}$$

where u_i and v_i denote the ith component of vectors $\mathbf{u}$ and $\mathbf{v}$, respectively.

3) z-Score: Consider a population with mean μ and standard deviation σ. Given an observation x, z-score gives the number of standard deviations the observed value is from the population mean

$$\texttt{zScore}(x; \mu, \sigma) = \frac{x - \mu}{\sigma} \tag{3}$$

Algorithm 1 Fingerprinting_at_GW($\mathcal{S}$, X)

Input: $\mathcal{S}$: Traffic session; X: device address
Variables: c: device class; p: prob. of classification, diffident_sessions[X]: list of sessions via interface X

1: **if** ExistsNewModel == TRUE **then**
2: $\mathcal{M} \leftarrow$ `From_Controller()` ▷ Update model
3: **end if**
4: $\mathbf{F} =$ `ExtractFeature`($\mathcal{S}$)
5: $[c, p] \leftarrow$ `Classify_GW`($\mathbf{F}, \mathcal{M}$)
6: **if** $p <$ threshold **then** ▷ If confidence is low
7: diffident_sessions[X].append($\mathbf{F}$)
8: **if** len(diffident_sessions[X] == θ **then**
9: `SendToController`(diffident_sessions[X])
10: diffident_sessions[X] = [] ▷ Empty list
11: **end if**
12: **return** NULL ▷ No classification done
13: **end if**
14: **return** c as class of $\mathcal{S}$

z-score is based on the mean and standard deviation, which have a breakdown point of 0%. Therefore, in scenarios where the data may have anomalies, it is common to replace these estimators (μ and σ) with their robust counterparts of median and median of all absolute deviations from the median (MAD). Both median and MAD have breakdown point of 50%. However, as we are currently generating data in a controlled environment, we find it sufficient to use z-score in this paper.

4) k-Means Clustering: k-means is a clustering algorithm used to partition a given set of data points into k clusters, where k is provided as input. If $\mu_1, \mu_2, \ldots, \mu_k$ represent the centroids of the k clusters $\mathcal{F}_k$'s, then the partition is achieved by minimizing the following objective function:

$$\sum_{i=1}^{k} \sum_{\mathbf{f} \in \mathcal{F}_k} ||\mathbf{f} - \mu_k||^2.$$

Seeded: k-means [7] is a semi-supervised clustering algorithm based on k-means, in which labeled data sets are used for seeding. These labeled data sets are used for initializing k-means in computing the centroids (instead of choosing random points). Thereafter, each observation $\mathbf{f}$ is assigned to the nearest cluster (i.e., the cluster with the nearest centroid to the observation), and the centroid of that cluster is recomputed. As will be described later in Section IV-C, our clustering algorithm uses a similar approach.

B. Classification at Gateways

Algorithm 1 lists the steps taken at a gateway, when it captures a traffic session $\mathcal{S}$. The first `if` statement checks if the controller has sent a new model back to this gateway, and if so, updates its model accordingly. In practice, this would be implemented separately and independent of the classification at gateways, using a publish–subscribe model, so that gateways receive the latest model at the earliest.

Line 4 in Algorithm 1 extracts features from traffic sessions. Supervised classification is performed in line 5, where $\mathcal{M}$ denotes the model used at gateways for classification. We assume the first such model is trained using a set of known devices, which we call as *seed devices*. The model itself is obtained from the controller, which retrains the model as and when necessary (explained later in Algorithm 2). All gateways use the same model $\mathcal{M}$ for classification.

The `Classify_GW` function runs a supervised classifier trained by the DEFT controller. Later in Section V, we discuss on the supervised learning algorithms evaluated for device classification. The `Classify_GW` function returns the device class c of the session, as well as a measure of confidence p for the class assigned. The second `if` statement assesses this confidence; if the confidence is low, the corresponding session's feature vector $\mathbf{F}$ is appended to the list *diffident_sessions*—a list of sessions predicted with low confidence.

Observe that *diffident_sessions* is indexed by X; X is the unique address of an IoT device, for example, its MAC address. This information allows the gateway to segregate sessions based on the device addresses, even while it is not certain of the device type. Note that, while a device can fake an MAC address it presents to the gateway, this does not pose a problem as long as the faked MAC address is not the same as another device connected to the gateway. Duplicate MAC addresses can be easily detected by a gateway. This segregation based on address is used only to aggregate feature vectors from the same device. When the number of low-confidence sessions observed from a particular device (i.e., length of list *diffident_sessions*[X]) is θ, it is sent over to the controller (line 9). Henceforth, we refer to *diffident_sessions*[X] as feature aggregate $\mathcal{F}$ (that is, $\mathcal{F}$ is a set of feature vectors coming from an unknown device).

C. Identifying New Devices at Controller

The `SendToController` function invoked by gateways (in Algorithm 1) sends feature aggregates to the controller. This function invokes Algorithm 2 at the controller. Consider the case where the controller has received a list of feature aggregates from some of its gateways. We denote this list as $\mathbb{P}$. Another important data structure is $\mathcal{R}$, which is a list of clusters from seed devices. A *cluster* here is a labeled feature aggregate, which has at least θ' number of feature vectors. We also maintain $\mathbb{T}$, a data structure similar to $\mathbb{P}$ (list of feature aggregates). $\mathbb{T}$ is used to reduce the feature aggregates in $\mathbb{P}$, such that each feature aggregate in $\mathbb{T}$ corresponds to a unique device type. That is, DEFT merges "similar" feature aggregates (potentially belong to same device type), as detailed in the algorithms below. Initially, $\mathbb{T}$ is bootstrapped with clusters from $\mathcal{R}$. The logic of having two related lists $\mathcal{R}$ and $\mathbb{T}$ is that, DEFT uses $\mathcal{R}$ to maintain the final clusters corresponding to device types.

For clustering, we first experimented with standard k-means algorithm. In this experiment, we used all devices listed in Table III and set k equal to the number of devices (16 here). We then evaluated the accuracy as the number of traffic sessions (feature vectors) correctly classified as belonging to the appropriate device. The accuracy achieved using k-means was a low 40.87%. Another disadvantage of this approach is that, we need to specify the number of devices as the input parameter k;

Algorithm 2 Clustering_at_Controller($\mathcal{P}$)

Input: $\mathbb{P}$: List of feature aggregates
Variables: $\mathbb{T}$: List of labeled feature aggregates
1: **for** each set $\mathcal{F} \in \mathbb{P}$ **do**
2: $\mathcal{G} =$ `nearestCluster`$(\mathbb{T}, \mathcal{F})$
3: **if** `withinICR`$(\mathcal{F}, \mathcal{G})$ == TRUE **then**
4: $\mathbb{T}$.`replace`$(\mathcal{G}, \mathcal{F} \cup \mathcal{G})$
5: **else**
6: $\mathbb{T}$.`add`$(\mathcal{F})$ ▷ add new cluster
7: **end if**
8: **end for**
9: `updateModelClusters`$(\mathbb{T})$

Algorithm 3 NearestCluster($\mathbb{T}, \mathcal{F}$)

Input: $\mathbb{T}$: List of labeled feature aggregates
$\mathcal{F}$: Set of feature vectors (points)
1: $\mathcal{C} = \zeta(\mathcal{F})$ ▷ computed using Eq. (1)
2: $\mathcal{G} = \arg\min_{\mathcal{X} \in \mathbb{T}}(C - \zeta(\mathcal{X}))$ ▷ find nearest based on centroid
3: **return** $\mathcal{G}$

note, this information is not available at the controller. To overcome these obstacles, we carried out analyses and made a few observations: 1) instead of clustering feature vectors, we can cluster feature aggregates and 2) intercluster distances is a good metric for clustering feature aggregates. The second observation comes from the analysis that we have provided in Section V-D3. Following these observations, we developed our clustering algorithm, which is inspired by the *seeded k*-means algorithm in [7]—a semi-supervised clustering technique. In this algorithm, the labels of the known data are not changed, but is used to estimate the centroids of the clusters. Based on this concept of using data with known labels, we proceed to develop our clustering algorithm given in Algorithm 2.

The basic idea here is to check if the feature aggregate obtained from a gateway is close to any of the known feature aggregate in $\mathbb{T}$. Each iteration in the `for` loop operates on one feature aggregate $\mathcal{F}$. We first find the cluster in $\mathbb{T}$ that is nearest to $\mathcal{F}$, using the function `nearestCluster` (line 2 of Algorithm 2). The `If` statement checks whether the distance to this nearest cluster $\mathcal{G}$ is within acceptable range, using the function `withinICR` (as described below, z-score is used for this purpose). In this case, the feature vectors in $\mathcal{F}$ is added to $\mathcal{G}$, and this new merged feature aggregate ($\mathcal{F} \cup \mathcal{G}$) replaces the old feature aggregate $\mathcal{G}$ in $\mathbb{T}$. If the nearest feature aggregate is not within the acceptable range, $\mathcal{F}$ is considered as a new feature aggregate in $\mathbb{T}$, and added to $\mathbb{T}$ (line 6). Finally, both the model $\mathcal{M}$ and the list of clusters in $\mathcal{R}$ are updated by invoking the function `updateModelClusters`.

Next, we describe the other algorithms used at DEFT the controller. The `nearestCluster` function is presented in Algorithm 3. The function computes the distances between centroid of the given feature aggregate $\mathcal{F}$ to the centroid of each feature aggregate in $\mathbb{T}$, and returns the feature aggregate to which the distance is minimum.

The function `withinICR`, defined in Algorithm 4, checks if the two given feature aggregates are within the range of the

Algorithm 4 WithinICR($\mathcal{F}_0, \mathcal{F}_1$)

Input: $\mathcal{F}_0, \mathcal{F}_1$: Feature aggregate
Variables: $\mathbb{T}$: List of sets of labeled feature vectors;
1: Compute centroids ζ_{F_0} and ζ_{F_1} using Eq. (1)
2: $D(\mathcal{F}_0, \mathcal{F}_1) =$ `dist`$(\zeta_{F_0}, \zeta_{F_1})$ ▷ computed using Eq. (2)
3: **if** | `zScore`$(D(\mathcal{F}_0, \mathcal{F}_1), \overline{\mu}, \overline{\sigma})| < \beta$ **then**
4: **return** TRUE
5: **end if**

Algorithm 5 EstimateClusterDist($\mathcal{R}$)

Input: $\mathcal{R}$: List of clusters used for training the classifier
1: $l =$ `len`$(\mathcal{R})$
2: **for** i in $[1..l-1]$ **do**
3: **for** j in $[i+1..l]$ **do**
4: $D[\mathcal{R}[i], \mathcal{R}[j]] =$ `dist`$(\mathcal{R}[i], \mathcal{R}[j])$
5: **end for**
6: **end for**
7: $\overline{\mu}, \overline{\sigma} =$ `mean_std`(D)

Algorithm 6 UpdateModelClusters($\mathcal{T}$)

Input: $\mathbb{T}$: List of feature aggregates
Variables: $\mathcal{R}$: List of clusters used for training the classifier; $\mathcal{M}$: Classifier model
1: Re-initialize $\mathcal{R}$ as an empty list
2: **for** each $\mathcal{F} \in \mathbb{T}$ **do**
3: **if** len$(\mathcal{F}) > \theta'$ **then**
4: $\mathcal{R}$.`add`$(\mathcal{F}'')$
5: **end if**
6: **end for**
7: $\mathcal{M} =$ `train`$(\mathcal{R})$ ▷ Retrain and obtain new model
8: `estimateClusterDist`$(\mathcal{R})$
9: `SendToGWs`$(\mathcal{M})$ ▷ Send new model to all gateways

intercluster distance measure. Given two feature aggregates, the algorithm computes the centroid of these feature aggregates (line 1) and the distance between the centroids (line 2). To decide whether these two feature aggregates can be merged or not, we do the following. Let $\overline{\mu}$ and $\overline{\sigma}$ denote the mean and standard deviation of the intercluster distances—distances between the clusters in $\mathcal{R}$. Estimation of $\overline{\mu}$ and $\overline{\sigma}$ are carried out in Algorithm 5. Given the estimated mean and standard deviation, and using the computed distance between the feature aggregates in line 2, we compute the z-score using (3). If the absolute value of z-score is less than a threshold β, DEFT considers the feature aggregates to be close enough to be merged.

Algorithm 5 estimates the mean and standard deviation of intercluster distances, and is self-explanatory. Algorithm 6 updates the cluster list $\mathcal{R}$, such that $\mathcal{R} \subset \mathbb{T}$ and each $\mathcal{F} \in \mathcal{R}$ has more than θ' feature vectors. These clusters are then used to update the model used for classification at the gateways; and this is achieved using the `train` function (line 7). The `train` function uses a supervised learning algorithm for training a classifier model. The specific classification algorithm deployed is given in the following section; however DEFT

only requires this classifier to be trained using a supervised approach.

D. Computational Complexity

In this section, we analyze the cost of fingerprinting at the gateways and the controller.

1) Gateway: The two main tasks at a gateway are: 1) feature extraction (line 4 in Algorithm 1) and 2) classification of the instance—traffic session here. Most well-known classifiers take time linear in the number of features for classifying a given instance (although for random forests classifier, we need to factor in the number of trees given as an input parameter). Since the number of features is small and fixed, the cost of classifying an instance can be seen as constant. Therefore, the computational cost at the gateway is dominated by the feature extraction process. An instance for classification is obtained by processing and extracting all relevant features from a traffic session. This essentially requires processing of each packet, with slightly different yet simple computations performed for extracting information for different features. For example, the number of packets, the statistical summaries of packet lengths, the duration of flow (connection), etc., are computed for different protocols. Therefore, feature extraction process takes $m \times \mathcal{O}(n)$, where m is the number of features and n is the number of packets in a session.

2) Controller: The two main operations at the DEFT controller are: a) clustering and b) update/retraining of model. For clustering, the computational cost due to the function `nearestCluster` is $\mathcal{O}(|\mathbb{T}|)$, and that due to `updateModelClusters` is also $\mathcal{O}(|\mathbb{T}|)$. Therefore, the clustering operation has a time complexity of $\mathcal{O}(|\mathbb{T}|)$. The cost of retraining the classifier depends on the supervised classification used. For example, the standard k-nearest neighbors (k-NNs) does not perform any computation for training. Whereas, naive Bayes classifier takes time linear in the product of the size of the training set and the feature dimension (m). Random forests with B trees can take between $\mathcal{O}(mB|F|\log^2|F|)$ and $\mathcal{O}(mB|F|^2\log|F|)$ time depending on the data (or, how balanced the trees are) [8]. Therefore, the computational time at the controller is dominated by the retraining process.

It is important to note here that, for both clustering and retraining of the classifier, the input is a list of feature aggregates; and a feature aggregate is essentially an aggregation of feature vectors (traffic sessions). Therefore, the costs of these operations are in terms of traffic sessions and, to highlight, not in packets. In addition, the feature aggregates that the controller receives are formed from only the diffident traffic sessions at the gateways (lines 8–11 of Algorithm 1). Therefore, the run-time cost of algorithms on the controller is practically low and acceptable. Besides, since the controller is hosted in the cloud, it can also be scaled up with additional resources as and when needed (say, when there is an increase in the rate of arrival of diffident sessions from gateways). Of course, the number of controllers can itself be increased when the number of gateways each controller manages increases beyond a limit—an important advantage of the hierarchical architecture of the DEFT system (as motivated in Section I).

TABLE III
INFORMATION ON IOT DEVICES USED

Label	Device	Brand	Sessions captured
0	Echo dot	Amazon	490
1	Smart remote	Broadlink	480
2	Camera (DCS930L)	D-Link	384
3	Camera (DCS5030L)	D-Link	410
4	Smart socket (DSPW215)	D-Link	672
5	Chromecast	Google	297
6	Home control	Google	529
7	Smart bulb	MI	295
8	Smart socket	Oittm	394
9	Hue light	Philips	644
10	Smart things	Samsung	587
11	Smart bulb (LB100)	TP-Link	482
12	Camera (NC250)	TP-Link	587
13	Camera (NC450)	TP-Link	494
14	Smart socket (HS100)	TP-Link	452
15	Smart socket (HS110)	TP-Link	387

V. PERFORMANCE EVALUATION

In the following, we carry out experiments to evaluate DEFT. Our design of DEFT works for both smart homes and enterprises; however, for the experiments here, we consider a network of smart homes.

A. Experimental Setup

Our experimental setup consists of 16 IoT devices relevant to smart homes. They are connected to the Internet via a gateway over Ethernet or Wi-Fi. We used Raspberry Pi 3 as a gateway, to capture traffic generated by the devices. We used an interval length of 15 min to define sessions. Table III gives the details of the devices and the number of sessions collected for each. In total, 7584 sessions were generated and captured over a period of seven days. In this controlled environment, MAC addresses in the packet headers were used to label the traffic sessions from different devices.

B. Scenario

The scenario we consider for evaluation consists of one controller and five gateways. Each gateway is located at a smart home, to which five devices are connected. Each home has 1–2 devices that are used in other homes as well. At the start of the experiment, we assume there are five known devices, each located at different homes; that is, the traffic of these devices are available for training. Therefore, when the system initializes, there are five *seed clusters* whose labels are known. We refer to these known devices as *seed devices* and the remaining ones as *test devices*. The initial model is trained using feature vectors from these seed devices.

The labels of the test devices are not known, and their traffic would be new to gateway(s) and controller. As described in the algorithms, when a gateway observes low classification probability of traffic session(s) from the new device, it sends the corresponding set of feature vectors to the controller. This scenario is executed offline, where traffic is captured and send as input to the gateway (Algorithm 1).

C. Metrics for Evaluation

Precision, recall, and F_1 are the commonly used metrics for multiclass classification. For a given class, precision, and recall are defined as

$$\text{precision} = \frac{\#\text{ True Positive}}{\#\text{ True Positive} + \#\text{ False Positive}}$$
$$\text{recall} = \frac{\#\text{ True Positive}}{\#\text{ True Positive} + \#\text{ False Negative}}.$$

Precision gives the fraction of correctly predicted instances of all those predicted for (and as) a particular class. Recall is the fraction of correctly predicted instances of the true instances of a class. For example, consider a set of instances for classification, where class A has 100 instances. If a classifier predicted 120 of all the instances to be of class A, but in reality only 90 were of class A (remaining 30 were wrongly predicted), then the precision for this class is 0.75. The recall for class A would be 0.9, since 90 of the 100 were "recalled."

Based on precision and recall, the F_1 score for a class is defined as

$$\text{F}_1 \text{ score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

The overall accuracy is the ratio of the sum of correctly predicted instances to the total number of instances.

D. Results

1) Centralized and Supervised Classification With Known Devices: In this section, we intend to evaluate the performance when fingerprinting is carried out in a centralized approach and using supervised classification with known and labeled IoT devices. We experimented such a scenario, in which training was performed using labeled data from all the 16 IoT devices listed in Table III. The testing phase had the same device set used for training, and no unknown IoT device. The following machine learning algorithms were tested: random forests, k-NN, and Gaussian and Bernoulli naive Bayes. All features listed in Table I were used. Among these algorithms, random forests classifier performed the best with 98% accuracy and naive Bayes performed worst with 85% accuracy. Observe that, these also represent the best results obtained in the scenario where all traffic sessions from all the gateways are sent to a centralized controller performing supervised classification. Since random forests classifier can also easily train in parallel as well as work on heterogeneous data types, for our evaluation of DEFT below, we used random forests for supervised classification at the gateways (in line 7 of Algorithm 6).

2) Clustering Accuracy: We first analyze the accuracy of clustering the data corresponding to test devices. Recall that a gateway sends a feature aggregate (set of points) to the controller when in doubt. The size of this set is controlled by a parameter θ (refer line 8 in Algorithm 1), we experiment the accuracy of clustering as a function of this parameter θ. We use all the features listed in Table I for this experiment.

If $\mathbb{P}$ denotes the list of feature aggregates received at the controller, let ν denote the number of merges required to reduce $\mathbb{P}$ to a set of clusters such that each cluster maps uniquely to its corresponding device. Let η denote the number of feature aggregates correctly merged to form the right clusters corresponding to devices

$$\text{Clustering accuracy} = \frac{\eta}{\nu}.$$

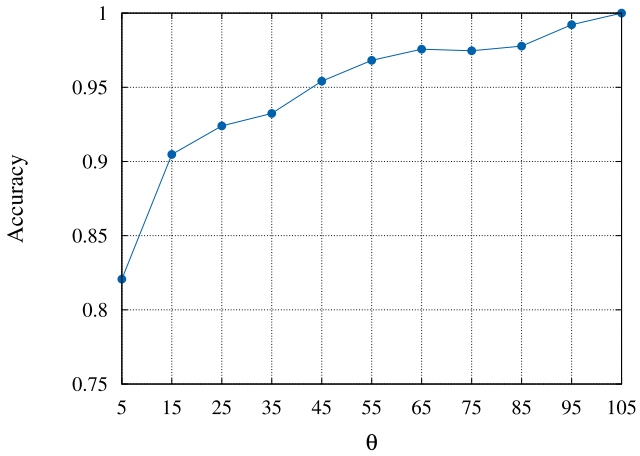

Fig. 5. Clustering accuracy as a function of θ.

Fig. 5 plots the clustering accuracy as a function of θ, the size of the feature aggregate. We observe that the accuracy is high even for small values of θ, and close to 100% accuracy is achieved with a feature aggregate of size 100. Note that, each point in the plot is the mean of clustering accuracies from five runs.

3) Intercluster and Intracluster Distances: Furthermore, we analyzed the intercluster distances and intracluster distances. These distances are essentially the Euclidean distances between the centroids of the corresponding clusters. Hence, intercluster distances for a device are obtained from the distances between its cluster and the clusters of the remaining devices. For intracluster distances of a device, we randomly partition the corresponding device cluster into ten equal parts, and calculate the distances between every pair. Fig. 6 depicts the box-plots for these distances. We observe almost an order of difference between the intracluster distances and the intercluster distances. This justifies the accuracies achieved due to clustering.

4) Analysis Based on z-Score: We analyzed clustering accuracy as a function of z-score, in Fig. 7. The feature vector dimension used is 100, and the cluster size is set to 100. We highlight that, the parameters for computing the z-score, namely $\overline{\mu}$ and $\overline{\sigma}$, are initially estimated from the data of seed devices. Subsequently, these parameters are continuously re-estimated as and when the clusters in $\mathcal{R}$ are updated. There is a range of values for the threshold β, approximately [2.2–3.0], for which the clustering accuracy peaks, and the accuracy decreases on both sides of this range. This is expected as a low z-score would create more number of clusters than necessary, and a high z-score will miss out creating new clusters for unknown test devices.

5) DEFT Classification Accuracy: Next, we analyze the accuracy of DEFT. The first five devices given in Table III were used as seed devices. More specifically, the list $\mathcal{R}$ was

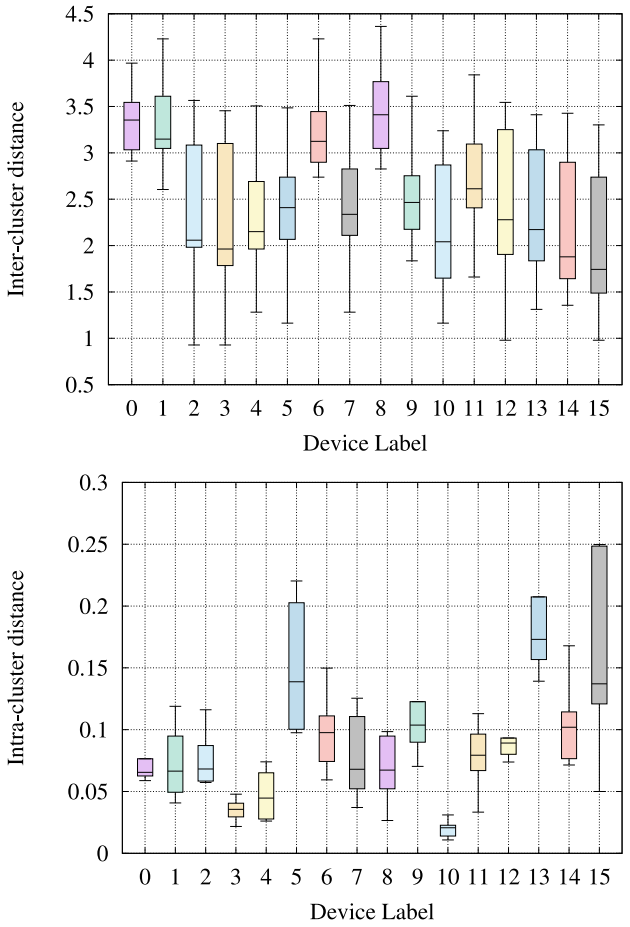

Fig. 6. Inter/Intra cluster box chart for each device.

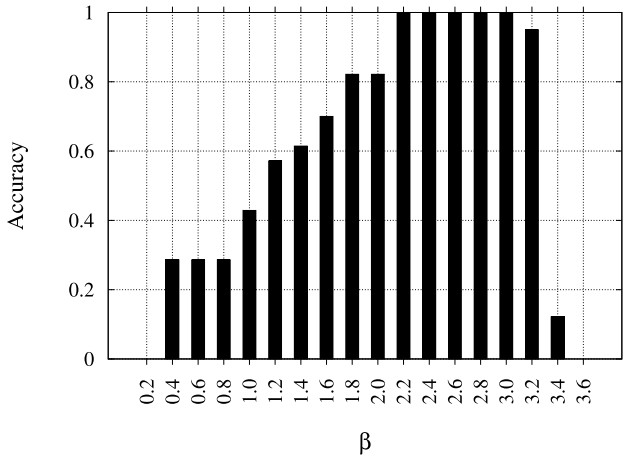

Fig. 7. Clustering accuracy as a function of β.

initialized with five seed clusters containing 50% of the total number of sessions collected for these device. That also means, no initial labeled dataset was used for the remaining 11 devices in DEFT. Therefore, clusters corresponding to these 11 devices were formed automatically and dynamically by DEFT during the experiment. The performance metrics we present are for all the devices, including the seed devices. We point out that, DEFT performs similarly, when initialized with other (randomly selected) sets of seed clusters.

TABLE IV
PRECISION, RECALL, AND F_1 SCORE FOR DEVICES

Device	Precision	Recall	F_1 Score
Echo dot	0.99	1.00	0.99
Smart remote	1.00	1.00	1.00
Camera (DCS930L)	1.00	1.00	1.00
Camera (DCS5030L)	1.00	1.00	1.00
Smart socket (DSPW215)	0.96	0.98	0.97
Chromecast	1.00	1.00	1.00
Home control	1.00	1.00	1.00
Smart bulb	0.95	1.00	0.97
Smart socket	1.00	1.00	1.00
Hue light	1.00	1.00	1.00
Smart things	1.00	1.00	1.00
Smart bulb (LB100)	1.00	0.99	1.00
Camera (NC250)	0.80	0.89	0.85
Camera (NC450)	1.00	1.00	1.00
Smart socket (HS100)	0.99	1.00	1.00
Smart socket (HS110)	0.90	0.66	0.76

Though fingerprint classification is performed at the gateway, the accuracy is dependent on the clustering accuracy. For this experiment, the cluster size θ was set to 105. The value of β for the z-score test was set to 3. We achieved an accuracy of approximately 97% for fingerprinting using the DEFT system. This high accuracy, obtained when we used all features listed in Table I, is close to that obtained with the centralized and supervised classification using labels of all IoT devices (presented in Section V-D1). Table IV gives the precision, recall, and F_1 score of all devices. Observe that, DEFT performs very good even in the presence of similar devices. For instance, the last four devices are from one vendor (TP-Link), and in particular two of them have same functionality but different models. In this case, the minimum F_1 score is still above 75%.

To understand how DEFT compares against a centralized and supervised learning approach for the same scenario used for the evaluation of DEFT, we conducted further experiments. In this setup, there were six labeled classes, five of which corresponded to five different IoT devices randomly picked in each experiment. The last class, *unknown*, represents another set of five IoT devices, again randomly selected. Therefore, we used a total of 10 devices for supervised training. Testing was performed using 16 devices, of which six were not used in training. This setting allows any new device unseen in the training phase to be classified as *unknown* device class, in addition to the five devices grouped into *unknown* class during the training phase. In other words, for these 11 devices, misclassification occurs only if they are classified into any class other than *unknown* class (i.e., into one of the first five known labeled classes). We used random forests algorithm as the supervised classifier. We carried out 50 trials of this experiment, selecting IoT devices randomly each time; and the mean accuracy achieved was 70.55%, with a 95% confidence interval of [67.2, 73.9]. Clearly, DEFT performs better than a centralized and supervised approach, under the same scenario wherein there are unknown devices to fingerprint.

6) Analysis of Feature Classes: In Fig. 8, we plot the overall accuracy for the different classes of feature vectors we

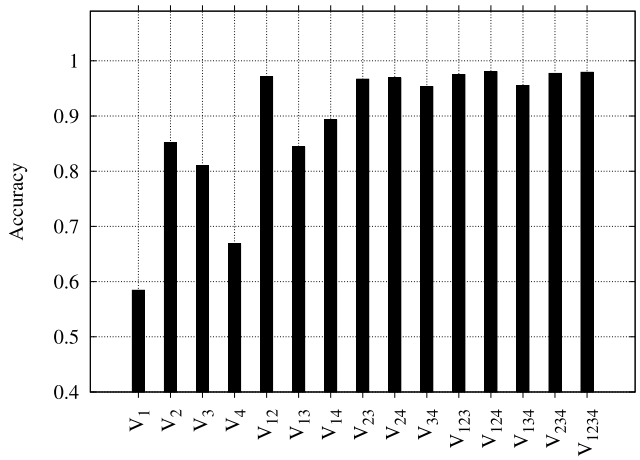

Fig. 8. Analysis of feature classes.

defined in Table I. Observing accuracies with the feature class V_2 and its combination with other classes (V_{12}, V_{23}, V_{24}), we note that this class of session-related features (which are protocol agnostic) contributes more to accurate fingerprinting than the rest. Though limited to the devices we tested with, we observe that for systems where only such aggregate and minimally privacy-intrusive features are available, fingerprinting using DEFT could still achieve high accuracy.

7) Scalability: Since the run-time complexity at the controller is dominated by the retraining of the classifier (Section IV-D), we estimated the time to build a classifier model. We utilized Python's scikit-learn library [9] for implementing the random forests classifier. On a Ubuntu-based VM, running on Intel Core i7-7700HQ@2.8 GHz with 4 cores and 4 GB of RAM, the controller took 81.79 ms to train a classifier using 7584 sessions across 16 devices. This indicates that the computation at the controller is not a bottleneck for scalability in the DEFT design. Moreover, we recall that, the controller performs retraining only when it receives a predefined number of feature aggregates from gateways.

To quantify the bandwidth saved due to the distributed approach of DEFT, we computed the sum of the sizes of sessions we gathered from all 16 IoT devices. The average session size was approximately 1.6 MB. Whereas, the average size of feature vectors (corresponding to these same sessions) was less than 550 bytes. As mentioned in Section I, a reinforcing classifier would need to analyze traffic continuously. In a centralized solution, since there is no intelligence besides the controller, all traffic would have to be sent to the controller. But DEFT does not send all feature vectors to the controller. Even if we make a conservative assumption that DEFT sends only one in ten feature vectors to the controller, we note that there is at least four orders of magnitude savings brought about by DEFT. This shows that DEFT is a highly scalable solution, in comparison with a centralized approach.

VI. Related Work

Fingerprinting, in general, refers to the specific characteristics that can be used identify the entity of interest. Network traffic at different layers can be used to fingerprint a few interesting entities, such as wireless devices (e.g., APs) [10]–[12], operating systems [13], applications [14], [15], etc. Depending on the goal, the features extracted are different. For example, fingerprinting has been used for indoor localization in recent works, based on information obtained from received signal strength and channel state information of communication links [16]–[18]. In another work [11], a sequence of interarrival times between packets flowing through an AP is used as features for fingerprinting APs. Whereas, the work in [13] extracted features from TCP/IP (such as IP TTL and TCP header values), TLS and HTTP protocols, for passive OS fingerprinting in the potential presence obfuscation strategies. Passive fingerprinting refers to the approach in which traffic is only monitored by the solution (p0f [19] is an example); as opposed to active approaches that send probes into the network for fingerprinting purposes (e.g., Nmap [20]). Our proposed solution DEFT passively monitors network traffic to fingerprint IoT devices.

As mentioned previously, IoT market presents a new set of challenges, primarily due to its scale, heterogeneity, and fast adoption. IoTs raise not only security concerns, but also privacy risks [21], [22]. Apthorpe *et al.* [23] demonstrated how sensitive private information can be inferred by analyzing network traffic from smart homes, even when the device traffic is encrypted. They show that user activities can be inferred from traffic rates of a few IoT devices. The work also discusses on device identification, analyzing features such as device MAC addresses, DNS queries, and traffic rates.

If fingerprints of devices turn out to be unique, then they can be used for authentication. Motivated by this objective, the work in [5], uses a multivariate Gaussian distribution to model fingerprint of a device; a supervised approach is used to learn the model. The work also applies transfer learning [24], to minimize the difference due to normal environmental changes. While this is an interesting proposal, due to lack of experimentation using real IoT devices, it is early to comment on the effectiveness of the proposal. We note that there is also scope in enhancing the model to make it a distributed solution. Additionally, it would likely be a new and challenging problem to extend the model to identify unknown devices.

A meta-classifier trained using supervised machine learning was developed in [25] for identifying IoT devices in two stages. In the first stage, IoT devices are differentiated from other devices; and in the second stage, identification of IoT devices is carried out. Besides extracting features from network, transport, and application layers, features are also extracted from data gathered from external sources (Alexa top website ranking [26] and geo-location of IP addresses). In another related work [27], a system for automatic identification of IoT devices and security enforcement of IoT devices is presented. For identification, the system proposed uses 23 features extracted from the fixed number of packets. A twofold classification method is proposed. For the first step, a supervised binary classifier is trained for each device. If multiple classifier gives positive results for a device, then an edit distance-based comparison is made in the second step to decide on the final device type. Each classifier is trained using

"one versus rest" data. This system uses fewer features than DEFT, and in particular does not use important features from DNS, TLS, HTTP, SSDP, STUN, etc. The disadvantage due to this was evident in the classification results—more than a third of the devices had an accuracy of only around 50%. Both the above-mentioned solutions use a centralized and supervised learning approach, which neither scales well nor identifies new device types.

IoT devices can also be identified by fingerprinting radio frequency signals transmitted over the air. A recent interesting work proposed in [28] uses symbolic aggregate approximation (SAX) to reduce a time-series representation of statistical features of RF signal. A supervised machine learning (k-NN) is applied after SAX reduction, and was observed to classify with high accuracy (in terms of the common metrics of accuracy, precision, recall, and F_1 score). Though this paper also uses a centralized and supervised learning approach, we find it complementary to this paper, as our solution does not analyze features below the network layer in the TCP/IP stack. In particular, it is possible to explore a distributed and semi-supervised version of the solution in [28] to enhance the capability of DEFT.

VII. Conclusion

In this paper, we developed DEFT, a distributed approach for fingerprinting IoT devices based on network traffic. Our design is scalable, and based on a hierarchical network architecture. The DEFT control logic, that can work as an application in SDN controller, develops, maintains, and deploys classifier model at all its gateways. The deployment can be done dynamically using the NFV and microservice architecture. The traffic is classified locally, without being sent to the controller; while the controller coordinates with its gateways to learn models for new unknown devices and updates the model dynamically. We evaluated DEFT on real-world IoT devices, and demonstrated its effectiveness in classifying IoT devices while also identifying new unknown devices. Our results also show that DEFT is scalable requiring significantly less (in several orders of magnitude) communication overhead toward the controller, in comparison with a centralized approach.

References

[1] *IoT Connections Outlook (Mobility Report)*, Ericsson, Stockholm, Sweden, Nov. 2017. Accessed: May 2018. [Online]. Available: https://www.ericsson.com/en/mobility-report/reports/november-2017/internet-of-things-outlook

[2] N. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic," presented at the Workshop Data (DAT), 2016.

[3] M. Antonakakis *et al.*, "Understanding the Mirai Botnet," in *Proc. 26th USENIX Security Symp.*, 2017, pp. 1093–1110.

[4] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial Internet of Things," in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2015, pp. 1–6.

[5] Y. Sharaf-Dabbagh and W. Saad, "On the authentication of devices in the Internet of Things," in *Proc. IEEE 17th Int. Symp. World Wireless Mobile Multimedia Netw. (WoWMoM)*, 2016, pp. 1–3.

[6] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Disc. Data Min. (KDD)*, 2017, pp. 1723–1732.

[7] S. Basu, A. Banerjee, and R. J. Mooney, "Semi-supervised clustering by seeding," in *Proc. 19th Int. Conf. Mach. Learn. (ICML)*, 2002, pp. 27–34.

[8] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artif. Intell.*, vol. 206, pp. 79–111, Jan. 2014.

[9] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Feb. 2011.

[10] S. Jana and S. K. Kasera, "On fast and accurate detection of unauthorized wireless access points using clock skews," *IEEE Trans. Mobile Comput.*, vol. 9, no. 3, pp. 449–462, Mar. 2010.

[11] K. Gao, C. Corbett, and R. Beyah, "A passive approach to wireless device fingerprinting," in *Proc. IEEE IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, 2010, pp. 383–392.

[12] Q. Xu, R. Zheng, W. Saad, and Z. Han, "Device fingerprinting in wireless networks: Challenges and opportunities," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 94–104, 1st Quart., 2016.

[13] B. Anderson and D. McGrew, "OS fingerprinting: New techniques and a study of information gain and obfuscation," in *Proc. IEEE Conf. Commun. Netw. Security (CNS)*, Oct. 2017, pp. 1–9.

[14] M. Korczyński and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, Apr. 2014, pp. 781–789.

[15] R. Bortolameotti *et al.*, "DECANTeR: Detection of anomalous outbound HTTP traffic by passive application fingerprinting," in *Proc.33rd Annu. Comput. Security Appl. Conf. (ACSAC)*, 2017, pp. 373–386.

[16] P. Fonseka and K. Sandrasegaran, "Indoor localization for IoT applications using fingerprinting," in *Proc. IEEE 4th World Forum Internet Things (WF-IoT)*, Feb. 2018, pp. 736–741.

[17] X. Wang, L. Gao, and S. Mao, "CSI phase fingerprinting for indoor localization with a deep learning approach," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1113–1123, Dec. 2016.

[18] Q. Song, S. Guo, X. Liu, and Y. Yang, "CSI amplitude fingerprinting-based NB-IoT indoor localization," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1494–1504, Jun. 2018.

[19] *p0f: A Tool for Passive Traffic Fingerprinting*. Accessed: May 2018. [Online]. Available: http://lcamtuf.coredump.cx/p0f3

[20] *Nmap: The Network Mapper—Free Security Scanner*. Accessed: May 2018. [Online]. Available: https://nmap.org

[21] M. R. Schurgot, D. A. Shinberg, and L. G. Greenwald, "Experiments with security and privacy in IoT networks," in *Proc. 16th IEEE Int. Symp. World Wireless Mobile Multimedia Netw. (WoWMoM)*, 2015, pp. 1–6.

[22] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Comput. Netw.*, vol. 76, pp. 146–164, Jan. 2015.

[23] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted IoT traffic," *CoRR*, vol. abs/1708.05044, 2017. [Online]. Available: http://arxiv.org/abs/1708.05044

[24] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[25] Y. Meidan *et al.*, "ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis," in *Proc. Symp. Appl. Comput. (SAC)*, 2017, pp. 506–509.

[26] *Alexa Top Sites*. Accessed: May 2018. [Online]. Available: https://www.alexa.com/topsites

[27] M. Miettinen *et al.*, "IoT sentinel demo: Automated device-type identification for security enforcement in IoT," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, 2017, pp. 2177–2184.

[28] G. Baldini, R. Giuliani, G. Steri, I. Sanchez, and C. Gentile, "The application of the symbolic aggregate approximation algorithm (SAX) to radio frequency fingerprinting of IoT devices," in *Proc. IEEE Symp. Commun. Veh. Technol. (SCVT)*, 2017, pp. 1–6.

Vijayanand Thangavelu received the B.E. degree in electronics and communications from the PSG College of Technology, Coimbatore, India, in 2015. He is currently pursuing the M.Sc. degree in electrical and computer engineering at the National University of Singapore, Singapore.

He joined Cisco Systems, Bangaluru, India, in 2015, where he was a Developer with the Security Business Unit for two years. His current research interests include networking, IoT security, and SDN/NFV-based architecture for IoT security.

Dinil Mon Divakaran (SM'11) received the master's degree from the Indian Institute of Technology (IIT) Madras, Chennai, India, and the Ph.D. degree from ENS Lyon, Lyon, France.

He was a Scientist and the Deputy Head of the Network Security Department, A*STAR Institute for Infocomm Research, Singapore. He was also an Assistant Professor with the School of Computing and Electrical Engineering, IIT Mandi, Mandi, India, and a Research Fellow with the National University of Singapore, Singapore. He currently leads research directions in network security within the Cyber Security Research and Development Division, Singapore Telecommunications (Singtel), Singapore. His current research interests include network security and computer networks.

Rishi Sairam received the Bachelor of Engineering degree from the Thiagarajar College of Engineering, Madurai, India, in 2015 and the Master of Science degree from the National University of Singapore, Singapore, in 2018.

He has approximately two years of industrial experience from Nokia Networks, Chennai, India, in the area of fixed networks. His current research interests include IoT, network security, and SDN/NFV. He is currently involved in the development of a framework for improving IoT security at the network edge.

Suman Sankar Bhunia received the M.Tech. and Ph.D. degrees from Jadavpur University, Kolkata, India, in 2010 and 2016, respectively.

He was a Researcher with the Singapore University of Technology and Design, Singapore, and the National University of Singapore, Singapore. His current research interests include IoT, ubiquitous computing, cyber security, software-defined networking, network function virtualization, and blockchain.

Mohan Gurusamy (M'00–SM'07) received the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Madras, Chennai, India, in 2000.

He joined the National University of Singapore, Singapore, in 2000, where he is currently an Associate Professor with the Department of Electrical and Computer Engineering. He has authored or co-authored over 200 publications including 2 books and 3 book chapters in the area of optical networks. His current research interests include software-defined networks, network function virtualization, IoT, cloud data center networks, and optical networks.

Dr. Gurusamy is currently serving as an Editor for the IEEE TRANSACTIONS ON CLOUD COMPUTING, *Computer Networks* (Elsevier), and *Photonic Network Communications* (Springer). He has served as the Lead Guest Editor for two special issues of *IEEE Communications Magazine* (OCS) in 2005 and a Co-Guest Editor for a special issue of *Optical Switching and Networking* (Elsevier) in 2008. He is serving or has served as a TPC Co-Chair for several conferences including IEEE GLOBECOM 2019 (ONS) and IEEE ICC 2008 (ONS).