Pinpointing Hidden IoT Devices via Spatial-temporal Traffic Fingerprinting

Xiaobo Ma*†, Jian Qu*†, Jianfeng Li*‖, John C.S. Lui‡, Zhenhua Li§, Xiaohong Guan*†
*MOE Key Lab for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an, China
†Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China
‡Department of Computer Science & Engineering, The Chinese University of Hong Kong, Hong Kong
§School of Software, Tsinghua University, Beijing, China
‖Department of Computing, The Hong Kong Polytechnic University, Hong Kong
Email: xma.cs@xjtu.edu.cn, qj904154277@stu.xjtu.edu.cn, jfli@sei.xjtu.edu.cn, cslui@cse.cuhk.edu.hk, lizhenhua1983@tsinghua.edu.cn, xhguan@xjtu.edu.cn

***Abstract*—With the popularization of Internet of Things (IoT) devices in smart home and industry fields, a huge number of IoT devices are connected to the Internet. However, what devices are connected to a network may not be known by the Internet Service Provider (ISP), since many IoT devices are placed within small networks (e.g., home networks) and are hidden behind network address translation (NAT). Without pinpointing IoT devices in a network, it is unlikely for the ISP to appropriately configure security policies and effectively manage the network. In this paper, we design an efficient and scalable system via spatial-temporal traffic fingerprinting. Our system can accurately identify typical IoT devices in a network, with the additional capability of identifying what devices are hidden behind NAT and how many they are. Through extensive evaluation, we demonstrate that the system can generally identify IoT devices with an F-Score above 0.999, and estimate the number of the same type of IoT device behind NAT with an average error below 5%. We also perform small-scale (labor-intensive) experiments to show that our system is promising in detecting user-IoT interactions.**

I. Introduction

There is an increasing number of Internet of Things (IoT) devices in the world. The total installed base was 15.4 billion in 2015, and is expected to be 30.7 billion in 2020 and 75.4 billion in 2025 [1]. Because of their sheer volume and weak security, IoT devices have become the target of hackers. In recent years, the rapid development of botnets for IoT devices like Mirai and Turii has caused great security risks [2], [3]. As reported, a botnet of IoT devices can even launch coordinated attacks to bring down a power grid [4]. After the vulnerability of an IoT device is discovered, it usually takes a longer time to fix it than that of personal computers. The updates of IoT devices require the user's consent, but many users do not pay much attention to these update alerts. This leads to the continual operation of many vulnerabilities in IoT devices, and attackers may penetrate the network through these vulnerabilities.

The huge number of IoT devices has also brought a significant problem. That is, the Internet Service Provider (ISP) finds it difficult to determine what IoT devices are connected within its administrative domain. Without pinpointing IoT devices in a network, it is unlikely for ISPs to perform security maintenance and effectively manage the network, e.g., appropriately configuring personalized security policies according to IoT devices' vulnerabilities, and allocating security resources based on IoT devices' population distribution.

Detecting IoT devices can be achieved in two approaches. One is actively probing and identifying the services that IoT devices open to the Internet. The other is passively fingerprinting the traffic of IoT devices. The active probing is currently adopted by many security companies and researchers. However, this approach has limited visibility of IoT devices in at least two aspects. First, most IoT devices are placed within small networks (e.g., home networks) and are hidden behind network address translation (NAT). Thus, their services may only be visible within a subnet and are completely *invisible* outside the network. Second, although some IoT devices may use globally-routable IP addresses and are visible outside their residing networks, their open services may not contain sufficient information that can facilitate unique identification. Although the active probing enables ISPs to probe IoT devices outside their managed networks, this is not their first priority.

In contrast, the passive approach overcomes the limitations of the active probing because it can observe the traffic from/to all IoT devices. Fig. 1 depicts the scenario of passively detecting IoT devices from an ISP's perspective. An ISP accommodates network services for numerous local networks. Each local network (e.g., home network) connects to the ISP via routers/gateways. Within a local network, different devices (e.g., computers, laptops, and IoT devices) connect to the local routers/gateways for accessing the Internet. Our aim is to perform *traffic fingerprinting* between the ISP and local networks so to answer the following research questions (RQ).

RQ1. Is it possible to extract IoT traffic features and train a detection model to identify a given type of IoT device from an ISP's perspective?

RQ2. When multiple IoT devices of the same type are deployed behind the same IP address (i.e., NAT), how to determine the number of such IoT devices?

Answering the above research questions is not easy. The major challenges are two-fold. The first challenge is learning-testing asymmetry. Specifically, one can collect (pure) traffic traces as training samples and learn the traffic patterns of an IoT device in a (clean) controlled testbed. However, when ISPs

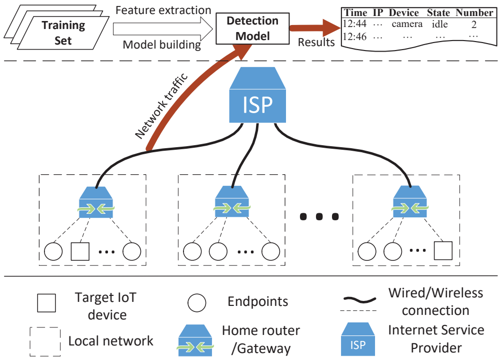


Fig. 1. An ISP's perspective of passively detecting IoT devices.

TABLE I
IoT DEVICES UNDER INVESTIGATION (YoM : YEAR OF MANUFACTURE).

Device Name (Abbr.)	Function	Manufacturer	YoM
DuSmart Speaker (DS)	Speaker	Baidu	2018
Mijia Camera SXJ02ZM (MC)	Camera	Xiaomi	2018
Lecoo Camera R1 (LC)	Camera	Sharetronic	2018
Ezviz Plug T30 (EP)	Plug	Espressif	2017
Amazon Echo (AE)	Speaker	Amazon	2014
DingDong Smart Speaker (DP)	Speaker	Grandway	2016
Mijia Plug (MP)	Plug	Xiaomi	2017
Mi AI Smart Speaker (MS)	Speaker	Xiaomi	2017
Dropcam (DC)	Camera	Dropcam	2013
HP Envy Printer (HP)	Printer	HP	2013
Netatmo Weather Station (NS)	Weather	Netatmo	2012
Netatmo Welcome (NW)	Camera	Netatmo	2012
PIX-STAR Photo-frame (PP)	Photo-frame	AMPAK	2015
Samsung SmartCam (SS)	Camera	Samsung	2014
Smart Things (ST)	Hub	Samsung	2013
Day Night Cloud Camera (DN)	Camera	TP-LINK	2015
Smart Plug (SP)	Plug	TP-LINK	2016

subsequently test the learned traffic patterns in their managed networks for detecting IoT devices, they may not be able to extract (pure) traffic traces as testing samples. The reason is that many devices are deployed behind NAT, making all devices (e.g., PCs and smartphones) behind NAT share the same IP address. The extracted traffic traces associated with such an IP address then become a *mixture* of traffic traces produced by different devices. Therefore, ISPs have no idea which traces constitute a pure testing sample. The second challenge is that simple features like external IP addresses that IoT devices contact are not reliable because they may change across networks due to content delivery network (CDN). Moreover, these simple features may not work in detection as many different IoT devices use the same cloud services.

To address these challenges, we design a system for detecting typical IoT devices via *spatial-temporal traffic fingerprinting*. The basic idea is to automatically extract short-term common subsequences of packet arrivals (i.e., sequence profiles) that appear relatively frequent, and meanwhile learn the long-term appearance relationships of all the extracted sequence profiles through the convolutional neural network (CNN). The sequence profiles hierarchically abstract specific packet-level features, and can describe the spatial context of packets (i.e., which packets with abstracted features frequently coexist with each other in a certain sequence). Incorporating the contextual information with hierarchically abstracted features enables our system to be accurate in pinpointing IoT devices even in a complicated network environment such as NAT.

To our best knowledge, we are the *first* to passively detect hidden IoT devices from an ISP perspective. Our contributions:

- We design an efficient and scalable system for IoT detection via spatial-temporal traffic fingerprinting. Our system can accurately identify typical IoT devices in a network, with the additional capability of identifying what devices are hidden behind NAT and how many they are.
- Our system can hierarchically extract spatial-temporal features of the traffic between IoT devices and their servers automatically. It has a low detection time complexity (i.e., almost linear to the number of packets) and works in an online fashion, thereby scalable to large networks and identifying IoT devices usually in just a few minutes.
- Through extensive evaluation in a network with nearly 3,000 users, we demonstrate that our system can generally detect IoT devices with an F-Score above 0.999, and estimate the number of the same type of IoT device behind NAT with an average error below 5%. We also detect several user-IoT interactions and obtain promising results.

Roadmap. In Sec. II, we present IoT traffic characteristics. Sec. III elaborates system design and Sec. IV evaluates it. We review the literature in Sec. V and conclude in Sec. VI.

II. UNDERSTANDING IoT DEVICES AND THEIR TRAFFIC CHARACTERISTICS

To understand the behavior of IoT devices, we collect and analyze the traffic traces of 17 typical IoT devices. The traffic traces come from two sources. One is via our testbed where we capture the traffic traces of 8 IoT devices. The other is a public dataset published by the University of New South Wales [5], offering us the traffic traces of 9 additional IoT devices. We detail these IoT devices' names, functions, manufactures, and year of manufacture (YoM) in Table I. Note that all IoT devices under investigation are Internet-enabled through WiFi.

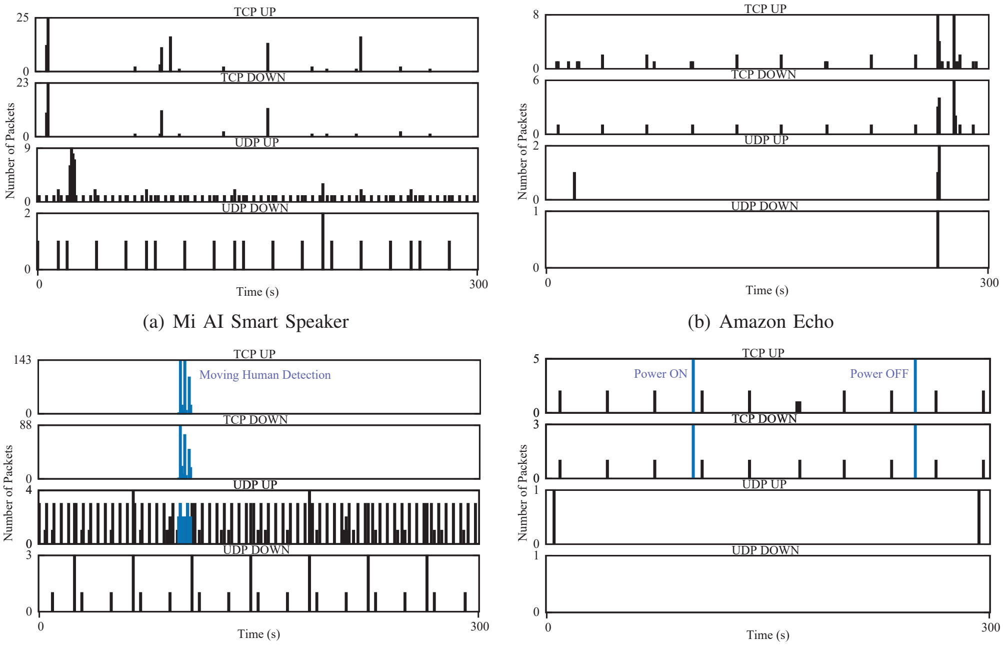
A. Three States of Typical IoT Devices

IoT devices, once turned on, have three possible states, i.e., initialization, idle, active. Their states are used to categorize the underlying types of activities of an (online) IoT device.

1) Initialization: Before using a new IoT device, one needs to configure the parameters of the device, and the configurations depend on the specific function. A new IoT device then enters the initialization state upon the first time it connects to the server. Some major tasks of such initialization include WiFi settings, authentication, and app-device binding.

2) Idle: At the idle state, no functional task is executed by the device, but heartbeat traffic takes place over a persistent connection between the device and the server. Consequently, most devices remain sending and receiving packets which constitute the *idle-state traffic*.

3) Active: When one interacts with an IoT device (e.g., voices and videos control, commands from mobile apps), the IoT device is updating its firmware, or a scheduled-task comes to execution, the device is in an active state and produces



(a) Mi AI Smart Speaker

(b) Amazon Echo

(c) Mijia Camera

(d) Ezviz Plug

Fig. 2. The number of packets varies over time for four IoT devices in the view of four filters (i.e., TCP UP, TCP DOWN, UDP UP, and UDP DOWN).

active-state traffic. A device is active when it is running certain tasks exclusive of those during initialization. For example, when one interacts with a smart speaker via voices, the smart speaker responds to voice commands and thus becomes active; Amazon Echo is considered active during the process of receiving and executing the commands from the mobile app.

B. IoT Traffic Characteristics

As different tasks are executed, an IoT device transits from one state to another. Note that the initialization state is transient, the idle state is the default state, and the active state only appears when a device is executing tasks. Therefore, one can expect that 1) the idle-state traffic is persistent; 2) the active-state traffic is abrupt and task-dependent (i.e., the characteristics may vary for executing different tasks). Throughout the whole life cycle of an IoT device, the initialization state may appear only a couple of times, resulting in pretty sparse traffic samples available in real-world network monitoring. Therefore, we focus on the idle-state/active-state traffic characteristics.

The idle-state traffic is attributed to the persistent connection between the IoT device and the server. When a user remotely controls the device, he/she will issue commands from his/her mobile phone. The commands are relayed by the server through the persistent connection to the device. The persistent connections of IoT devices differ from each other in terms of packet arrivals. In Fig. 2, we demonstrate how the number of packets extracted by four filters (i.e., TCP UP, TCP DOWN, UDP UP, and UDP DOWN) varies over time, where the width and the height of each bar denote one second epoch and the number of packets in that epoch, respectively. We observe that the four devices exhibit different packet arrival patterns, but all patterns comprise regular and irregular ones.

Fig. 2(a) and Fig. 2(b) depict the idle-state traffic pattern of Mi AI Smart Speaker and Amazon Echo, respectively. It can be observed that the former produces significant TCP and UDP traffic, while the latter primarily produces TCP traffic, in both directions. Despite being idle, both devices have significant irregular packet arrivals under certain filters (e.g., UDP traffic of Amazon Echo, TCP traffic of Mi AI Smart Speaker).

When user activities are triggered, the idle-state traffic will be interleaved with the active-state traffic. Fig. 2(c) and Fig. 2(d) depict the idle-state traffic with the active-state traffic of Mijia Camera and Ezviz Plug, respectively. We see that the two devices mainly produce TCP and UDP traffic, respectively. The blue bars represent the traffic produced by user activities. Such occasional occurrences of user activities (e.g., Moving Human Detection and Power ON/OFF) result in the idle-state traffic patterns being temporarily interfered.

Besides packet arrivals that characterize IoT traffic from the temporal perspective, we also use protocol-specific features encapsulated in packet headers, such as protocol type and packet length, to provide spatial information to understand IoT traffic characteristics. Fig. 3 is a Sankey diagram of Amazon Echo traffic (42,818 packets collected in 24 hours). The diagram contains statistics of protocols, addresses, ports, etc. We see that Amazon Echo communicates with diverse protocols, a large number of local/external server IP addresses, and distinct ports offering various services. Protocol-specific features enable us to distinguish between the same type of IoT devices of different firmware versions. For example, DuSmart Speaker produced in Apr. 2019, compared to that in Sep. 2018, incorporates Simple Service Discovery Protocol and reduces the request frequency of Network Time Protocol.

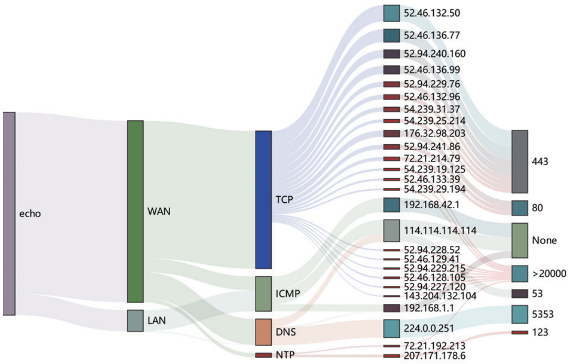


Fig. 3. Sankey diagram of Amazon echo traffic. Bars from left to right represent device name, WAN or LAN, protocol, server IP address, and port. "None" means that the protocol ICMP has no port.

C. Challenges for Bridging Characteristics and Detection

The IoT traffic characteristics presented above reveal that (temporal) packet arrivals and (spatial) protocol-specific features contain rich information in support of identifying the presence of IoT devices and distinguishing between different types of IoT devices. At first glance, exploring certain *significant* and *regular* traffic characteristics to detect IoT devices may be an obvious approach. For instance, one could easily leverage the UDP DOWN patterns of Mi AI Smart Speaker and Mijia Camera demonstrated in Fig. 2(a) and Fig. 2(c) via spectrum analysis of periodic signals. However, such patterns are not necessarily unique, thereby resulting in false positives.

To reduce false positives, one may correlate the patterns in the view of the four filters in Fig. 2. Unfortunately, an internal IP address, which may represent an IoT device, a non-IoT device, or many devices behind NAT, may contact numerous external server IP addresses. One has no prior knowledge of which internal IP addresses host only one device and which ones host more than one device. Accordingly, whether the external server IP addresses that an internal IP address contacts, in whole or in part, serve for a certain IoT device is agnostic. Therefore, grouping the traffic associated with an internal IP address in the view of the four filters becomes challenging.

One may further combine the observed set of external server IP addresses that an IoT device contacts in a controlled testbed so to reduce the combinational space. Nevertheless, matching the set external server IP addresses may necessitate observing the inbound-outbound traffic in a monitored network until most external server IP addresses are contacted. Although achieving this may not be time-consuming in a controlled testbed, observing most external server IP addresses in a monitored network requires a real-life user to trigger all related functions of an IoT device, which may take a long period of time (e.g., 24 hours). This long period of time severely limits the timeliness of IoT detection, not to mention that it introduces additional storage cost. Worse still, a long period of time of observation may not result in successfully matching the set of external server IP addresses in the presence of CDN.

The detection becomes more challenging with the prevalence of public services. An increasing number of IoT devices use public services, such as `https://api.amazon.com/`. These public services are also simultaneously used by many other applications. Even in the case of a private service, a series of different IoT devices of the same manufacturer tend to use the same private service. The detection is further complicated by the fact that many IoT devices are connected behind NAT, drastically raising the detection complexity because all devices behind NAT produce traffic originated from and destined to the same IP address from an ISP's perspective.

TABLE II
SUMMARY OF MAJOR NOTATIONS.

Notation	Definition
v	feature vector extracted from one packet
$v_i(k)$	the kth element of feature vector i
B	packet burst
B_j^a	the jth feature vector in packet burst a
S	sequence profile
$N(S)$	the occurrence number of S
$V(S)$	the importance value of S
$L(\cdot)$	longest common subsequence function
$D(\cdot)$	distance function between two vectors
$I(\cdot)$	abstraction function between two vectors

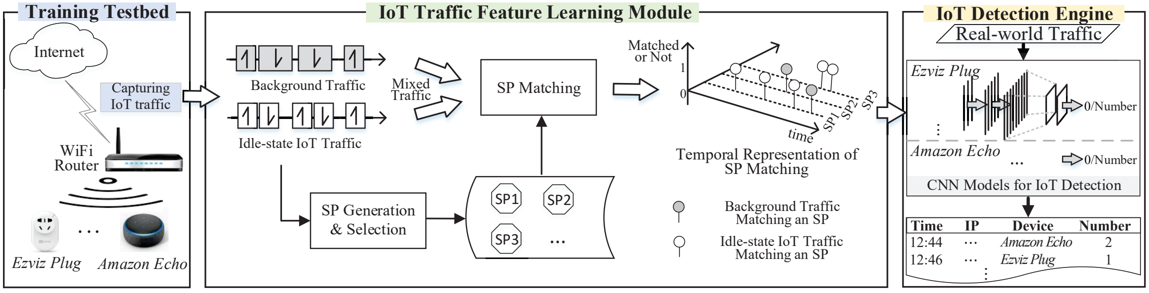
III. System Design

To tackle the challenges for bridging traffic characteristics and IoT detection, we use the following design objectives. First, a comprehensive approach that can characterize different IoT devices in a complicated network environment is required. Second, although an IoT device may occasionally have simple yet unique features such as external IP addresses (and domain names) that it contacts, when these features are not used, the system can also work well to ensure its general suitability to all types of devices. Third, to minimize training overhead, user intervention should be kept at a minimum when the system is learning the traffic characteristics of IoT devices.

Following the above design objectives, we propose the system architecture in Fig. 4. In training stage, we capture and label traffic for each IoT device. We keep IoT devices in idle state without user intervention, thereby making the collection and labeling of traffic for training highly automated. We then mix the collected idle-state traffic of each IoT device with the background traffic (consisting of non-IoT traffic and the traffic of the remaining IoT devices). Spatial-temporal features of the mixed traffic are extracted by the IoT traffic feature learning module. In detection stage, the IoT detection engine identifies IoT devices and estimates their numbers in real-world traffic. Table II lists major notations defined in our system.

A. Hierarchical Feature Extraction

IoT traffic exhibits significantly repeating patterns and some packet sequences occur frequently. Such packet sequences can be informative in IoT detection because they contain not only the information of individual packets, but also the short-term sequential structure of packet arrivals. Inspired by this observation, our system hierarchically extracts sequence profile (SP) as features from packet sequences as shown in Fig. 5.



Time	IP	Device	Number
12:44	···	Amazon Echo	2
12:46	···	Ezviz Plug	1

Fig. 4. The system architecture of detecting IoT devices and estimating their numbers (SP: Sequence Profile).

1) Packet Vectorization: For each packet, we only extract its header information as its application-layer payload may be encrypted and processing the payload is time-consuming. At the network layer, we extract fields in the IP header including Times to Live (TTL), total length, and protocol flag. The TTL value depends on networking system implementation of IoT devices, and could be a discriminative feature. Protocol flag indicates which transport protocol (i.e., UDP or TCP) is employed, and we transform it into a binary value. At the transport layer, we extract flags, window size, options, and payload length in the TCP header, and only payload length in the UDP header. Using these cross-layer header information, we represent each packet by a packet vector (PV). Table III summarizes PVs of different packets.

2) Packet Burst Extraction: The traffic between an IoT device and the server comprises packet bursts. Each packet burst results from a certain semantic network activity, such as time calibration. To capture such activities, we group packets into bursts. The time interval between two adjacent bursts should not be less than one second (i.e., Interval of Bursts in Table V). Packet bursts are denoted by $B^1, B^2, \ldots, B^n$. As shown in Fig. 5, each burst, say B^1, is sequentially represented by the PVs of packets (i.e., Packet 1 and Packet 2).

3) Sequence Profile Generation: We generate the longest common subsequence of two bursts as a Sequence Profile (SP). To extract such subsequences, we define the distance between two PVs. For two PVs of different protocols, we define their distance to be infinite. For two PVs pertaining to the same protocol, the distance is the summation of the distances between their counterpart elements. Formally, the distance between two PVs, v_i and v_j, of Packets i and j is

$$D\left(v_i, v_j\right) = \sum_{k=1}^{len(v_i)} \min\left\{w_k D_e\left(v_i(k), v_j(k)\right), m_k\right\}, \quad (1)$$

where $len(\cdot)$ calculates the vector length, w_k is the weight of the kth field of v_i (or equivalently v_j), $D_e\left(v_i(k), v_j(k)\right)$ is the distance between the counterpart elements of v_i and v_j, and m_k is the maximum value of $D_e\left(v_i(k), v_j(k)\right)$. For a field in a PV, the distance metric can be either binary or digital. If the distance metric of the kth field needs *exact matching*, we have

$$D_e\left(v_i(k), v_j(k)\right) = \begin{cases} 0, & v_i(k) \neq v_j(k), \\ 1, & v_i(k) = v_j(k). \end{cases} \quad (2)$$

Otherwise, we compute the *digital distance*

$$D_e\left(v_i(k), v_j(k)\right) = |v_i(k) - v_j(k)|. \quad (3)$$

After defining the distance between two PVs, we extract the longest common subsequence of two bursts, say B^a and B^b, based on dynamic programming [6], [7]. Assume that $B^a = (v_1^a, v_2^a, \ldots, v_{n_a}^a)$ and $B^b = (v_1^b, v_2^b, \ldots, v_{n_b}^b)$, with n_a and n_b PVs, respectively. Let B_i^a (resp. B_j^b) be the sequence consisting of the first i (resp. j) PVs of B^a (resp. B^b). We denote by $L\left(i, j\right)$ the longest common subsequence of B_i^a and B_j^b. Then, we have $L(n_a, n_b)$ represent the longest common subsequence of B^a and B^b. $L(n_a, n_b)$ can be recursively derived as follows

$$L\left(i, j\right) = \begin{cases} \emptyset, \quad \text{if } i = 0 \text{ or } j = 0, \\ L\left(i-1, j-1\right) \frown \left[I\left(v_i, v_j\right)\right], \\ \quad \text{if } i, j > 0 \text{ and } D\left(v_i, v_j\right) < d, \\ \max(L\left(i, j-1\right), L\left(i-1, j\right)), \\ \quad \text{if } i, j > 0 \text{ and } D(v_i, v_j) \geq d, \end{cases} \quad (4)$$

where $A \frown [b]$ means that b, as a new element, is added to the end of sequence A, d is a manually selected distance threshold discriminating the proximity of two PVs, and $I(v_i, v_j)$ is a function to compute an abstracted representation of two (similar) PVs v_i and v_j. Such an abstracted representation reflects whether the kth element of $I(v_i, v_j)$ is unique or could be "Any" value. Formally, the kth element of $I(v_i, v_j)$ is

$$I(v_i, v_j)(k) = \begin{cases} v_i(k), & v_i(k) = v_j(k), \\ \text{Any}, & v_i(k) \neq v_j(k). \end{cases} \quad (5)$$

For every two bursts B^a and B^b, we obtain one longest common subsequence $L(n_a, n_b)$ as an SP.

4) Sequence Profile Selection: For some IoT devices, such as smart plugs, there are usually fewer than 10 different SPs. For other devices, there may be hundreds of different SPs. If all the SPs are used as features, though not impossible, huge computation overhead would be introduced in both training and detection. Fortunately, according to measurement in Sec. IV, only a few informative SPs are needed to characterize an IoT device. Therefore, we only select informative SPs as features. At first glance, term frequency-inverse document frequency (TF-IDF), commonly used in NLP [8], can be a candidate selection method. However, TF-IDF is not suitable for our problem because SPs with high TF-IDF may be scarce in IoT traffic and cannot be used to estimate the number of IoT devices. The scarcity of SPs will also enlarge the

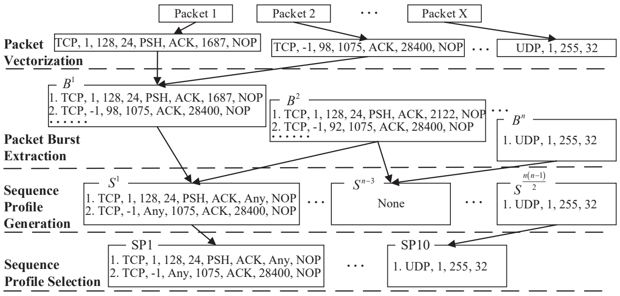

Packet 1
Packet 2
Packet X
Packet Vectorization
TCP, 1, 128, 24, PSH, ACK, 1687, NOP
TCP, -1, 98, 1075, ACK, 28400, NOP
UDP, 1, 255, 32
B^1
1. TCP, 1, 128, 24, PSH, ACK, 1687, NOP
2. TCP, -1, 98, 1075, ACK, 28400, NOP
B^2
1. TCP, 1, 128, 24, PSH, ACK, 2122, NOP
2. TCP, -1, 92, 1075, ACK, 28400, NOP
B^n
1. UDP, 1, 255, 32
Packet Burst Extraction
Sequence Profile Generation
S^1
1. TCP, 1, 128, 24, PSH, ACK, Any, NOP
2. TCP, -1, Any, 1075, ACK, 28400, NOP
S^{n-3}
None
$S^{\frac{n(n-1)}{2}}$
1. UDP, 1, 255, 32
Sequence Profile Selection
SP1
1. TCP, 1, 128, 24, PSH, ACK, Any, NOP
2. TCP, -1, Any, 1075, ACK, 28400, NOP
SP10
1. UDP, 1, 255, 32

Fig. 5. An example of generating SPs via hierarchical feature extraction.

detection time window. As an alternative method, we define the importance of an SP by combining its temporal frequency and spatial generalizability, i.e., frequency of occurrences in all SPs derived from every two bursts and the number of its elements with a value of "Any". Formally, we define the importance as

$$V(S) = \sqrt{N(S)} \times \sum_{j=1}^{len(S)} \sum_{i=1}^{len(v^j)} \delta\left(v_i^j\right), \tag{6}$$

where S is an SP, $N(S)$ is the frequency of occurrences of S, v^j is the jth element of S, v_i^j is the ith element of v^j, and $\delta(x)$ is an indication function

$$\delta(x) = \begin{cases} 0 & \text{if } x = \text{Any}, \\ 1 & \text{if } x \neq \text{Any}. \end{cases} \tag{7}$$

B. IoT Device Detection and Population Estimation

Our system makes use of SPs to detect IoT devices and estimate their population (i.e., the number of IoT devices of the same type). Individual SPs can only reflect the short-term sequential and spatial characteristics of a certain network activity. To characterize the long-term temporal patterns of network activities, we search in the mixed traffic, as shown in Fig. 4, for the (generated and selected) SPs. Then, a temporal representation of SP matching, in the form of a 3D (time–SPs–match or not) space, is generated. This representation incorporates the SPs that form different signal channels describing diverse aspects of IoT traffic, hence being able to characterize the spatial-temporal correlation across network activities.

To search for the SPs in the mixed traffic, we transform all packets in the mixed traffic into PVs, and match them against the SPs. For each SP, the output is a one-dimensional array describing the matching results along the time axis. The matching here is essentially the string subsequence matching. The only difference is that we need to count the number of SP matching based on the matched or not information. When one subsequence of PVs is matched with an SP, the number of SP matching will be increased by one at the time window where the subsequence begins. Fig. 6 shows examples of SP matching over DuSmart Speaker's traffic and background traffic.

By leveraging the SP matching representation, we choose CNN to detect IoT devices and estimate their population. The reason why we employ CNN is that it can automatically extract long-term patterns of individual SPs and spatial-temporal correlation across different SPs [9]–[11]. Our CNN model consists of 3 convolution layers, 3 pooling layers, 2 fully connected layers. The input of the first convolution layer is time arrays corresponding to the selected SPs. To avoid over-fitting of neural networks, we apply the dropout layers [12], [13]. During training, if we just detect whether the target device is in the traffic, we use the softmax and cross-entropy loss function. If we want to know the number of the target device, we use the Rectified Linear Unit (ReLU) and MSEloss function to calculate it, as is more computationally intensive. Therefore, to ensure the scalability of our system, we train two CNNs for each IoT device. One is for detection and works continuously. The other is for population estimation and is only launched upon successful detection of the target IoT device.

To train the above two CNNs, we need to build positive and negative samples with ground truth. Considering that IoT traffic may be interleaved by background traffic, we mix IoT traffic with background traffic as positive samples, and background traffic itself as negative samples. Denote IoT traffic by T_i, and background traffic by T_b. Then, training samples can be expressed as: $F_c(T_b) = 0$ and $F_c(T_b + n \otimes T_i) = 1$, where F_c is our CNN classifier for IoT device detection, and $n \otimes T_i$ denotes the superposition of the target device's traffic for n times. An output of 0 and 1 represent the absence and presence of the target device, respectively. To estimate the number of the target device, training samples are expressed as $F_e(T_b + n \otimes T_i) = n$, where n denotes the number of the target device, and F_e is the regression model for population estimation.

C. User-IoT Interaction Detection

We detect user-IoT interactions by labeling the active-state traffic for each action and representing the active-state traffic as SPs. Compared to inspecting IoT devices based on idle-state traffic without user intervention, collecting active-state traffic is labor-intensive since one needs to repeat the same type of interaction many times, especially those interactions relying on physical contact with IoT devices. For example, triggering moving human detection function of Mijia Camera requires people moving in front of the device. Second, a certain user-IoT interaction commonly induces a limited number (typically one) of packet bursts within a short period of time. To simplify the feature representation of such packet bursts, we extract SPs between every pair of packet bursts induced by repeating the user-IoT interaction many times, and in turn abstract these extracted SPs into one SP via dynamic programming in (4).

However, the computational complexity of dynamic programming increases drastically with the number of extracted SPs grows. We use an approximate algorithm to sequentially merge the extracted SPs. Specifically, the algorithm merges two SPs as a new SP by extracting their longest common subsequence, and recursively merges this new SP with more SPs until the final SP matches all packet bursts. The merging order is important. If we start by merging two random SPs, it is likely to lose useful information or even get an empty result. Instead, we start by merging SPs with high values of V defined in (6), which outperforms random merging.

IV. Evaluation

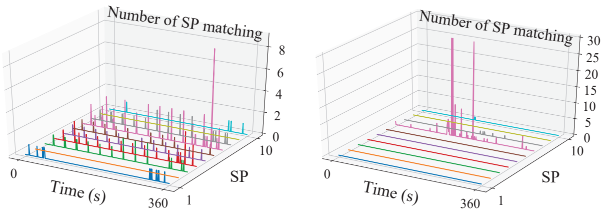
We evaluate the performance of our system in IoT device identification, population estimation, user-IoT interaction detection, and analyze its scalability in practical deployment.

A. Dataset Collection and Preprocessing

The data includes IoT traffic and background traffic. The IoT traffic was from our testbed and a public dataset published by the University of New South Wales [5]. The target IoT devices are listed in Table I. To further collect high-quality background traffic, we must ensure the statistical diversity of the background traffic. Moreover, the background traffic cannot contain any traffic of the target IoT devices. To fulfill these requirements, we *deliberately* build the background traffic using the traffic before the wide prosperity of IoT. Specifically, the background traffic, with a size of 1.1TB, was captured on the border of our campus network from Nov. 9th to Nov. 11th, 2015. It is associated with 2,952 unique IP addresses and all of them are distributed in students' apartments. It can be reasonably considered that the background traffic contains little traffic generated by the target IoT devices because 1) the target IoT devices produced after 2015 are impossible to appear in the background traffic collected in 2015 and 2) other devices are generally not used in students' apartments of our campus.

For each IoT device, we extract (idle-state) packet bursts, generate SPs, and select (informative) SPs to represent its traffic characteristics. The number of selected SPs is upper bounded by 10 (i.e., Maximum SP Number in Table V). Then, we match selected SPs against positive and negative traffic samples. The aim is to obtain time arrays corresponding to selected SPs, where each time array depicts how the number of SP matching (of a certain SP) varies over the time windows (i.e., CNN Time Window in Table V). These time arrays can be directly fed into CNN models for IoT training and detection.

Fig. 6 exemplifies time arrays of matching SPs of DuSmart Speaker against positive and negative traffic samples. We see that time arrays for positive samples are regular with occasional irregularity, while those for negative samples are abrupt and irregular. In this example, for ease of demonstration, we build positive samples using pure idle-state traffic of DuSmart Speaker, and negative samples using background traffic.



(a) DuSmart Speaker traffic (positive) (b) Background traffic (negative)

Fig. 6. Time arrays of matching SPs against positive/negative traffic samples.

In practice, positive and negative traffic samples for generating time arrays for an IoT device, say X, are built in two steps. First, we randomly select IoT devices other than X, blend their traces into background traffic, and get negative traffic samples. Then, we add the traces of X to negative traffic samples, and obtain positive traffic samples. Such steps fully consider that the traffic of a certain IoT device may be interleaved by that of all other devices. Tables IV and V list our parameter settings. Table IV is for calculating the distance between two PVs in (1), where Weight and Max mean w_k and m_k, respectively. Table V includes parameters including maximum SP number, distance threshold d, interval of bursts, and training parameters.

B. Results and Insights

1) IoT Device Detection: Fig. 7 shows precision and recall for all devices in Table I. We use 50GB background traffic in training and the rest in testing, and calculate precision and recall for the traffic traces within each 360-second time window. Fig. 7(a) is the results using our default features without the information of domain name, IP address and port, while Fig. 7(b) shows the results after adding these information into our system. We observe that the performance is not improved when domain name, IP address and port is added, implying that our system does not rely on these features.

Precision and recall of most devices are greater than 99.9%. Netatmo Weather Station and PIX-STAR Photo-frame have high precision but relatively low recall. This is because for these two devices the time interval between two consecutive packets is larger than the CNN time window. We can simply increase recall to 99% by increasing the CNN time window.

Both precision and recall increase as the detection time proceeds. Further, we want to know the minimum time for successfully detecting IoT devices. We test the detection response time for all devices (i.e., the time lag between an IoT device being connected and the successful detection) with the F-Score larger than 0.99. Table VI presents the detection response time rounded up to one minute. We see that most IoT devices can be detected in just a few minutes after they are connected.

Answer to RQ1: We profile each IoT device using idle-state traffic characteristics, and represent spatial-temporal features in a CNN-resolvable form. We can accurately detect IoT devices with F-Score above 0.999 in just a few minutes.

2) IoT Device Population Estimation: To perform population estimation (i.e., estimate the number of the same type of IoT devices behind NAT), we mix the device's traffic multiple times to simulate the scenario of multiple devices are behind NAT. Note that the range of the estimated number in testing is the same as that in training. If the actual number is outside this range, our system can still estimate a number. However, the error would be uncontrollable. We perform population estimation for all IoT devices with a maximum number of 100, and the average error is less than 5%. Fig. 8 shows the results of two particular IoT devices. The X-axis represents the actual number, and the Y-axis is the estimated number. We see that the two numbers are pretty close when we vary the actual number. For a certain device, our observation is that the error increases as the actual number grows. Estimation errors also differ across different devices. Normally, IoT devices with complicated traffic patterns tend to have large estimation errors.

TABLE III

FEATURES USED FOR DIFFERENT PROTOCOLS.

Protocol	Features
TCP	Time, Protocol, Direction, TTL, Payload Length, TCP Flags, TCP Window Size and TCP Options.
UDP	Time, Protocol, Direction, TTL and Payload Length.
Others	The same as UDP.

TABLE IV

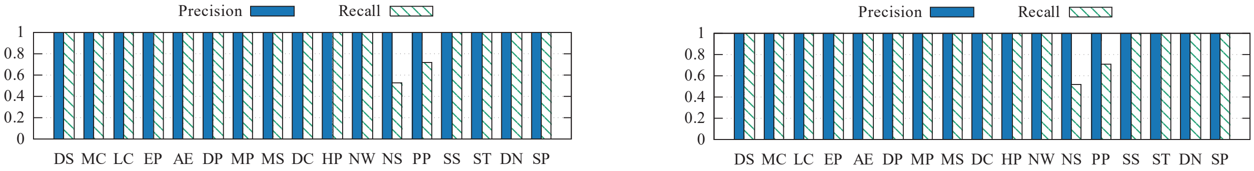
DISTANCE PARAMETERS BETWEEN TWO PVS.

Feature	Algorithm	Weight	Max
Direction	Exact Matching	4	4
TTL	Digital Distance	0.5	2
Payload Length	Exact Matching	2	2
TCP Flags	Exact Matching	2	2
TCP Window	Digital Distance	0.005	2
TCP Options	Exact Matching	1	1

TABLE V

TRAINING PARAMETER SETTINGS.

Parameters	Value
Maximum SP Number	10
Distance Threshold d	2.5
Interval of Bursts	1s
CNN Time Window	360s
CNN Leaning Rate	0.0005
Training Algorithm	Adam



(a) Without the information of domain name, IP address and port

(b) With the information of domain name, IP address and port

Fig. 7. Precision and recall of IoT device detection.

TABLE VI

DETECTION RESPONSE TIME FOR IOT DEVICES.

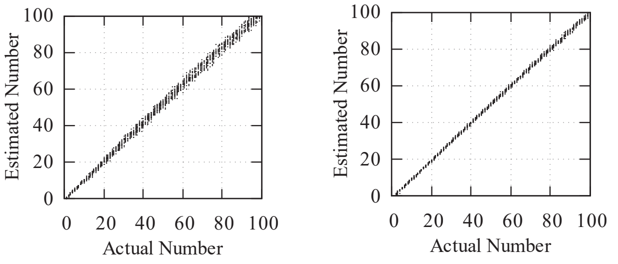
Device	Time (min)	Device	Time (min)	Device	Time (min)
DS	1	MC	1	LC	2
EP	2	AE	4	DP	1
MP	1	MS	2	DC	1
HP	2	NW	1	NS	13
PP(DN)	12(2)	SS(SP)	1(4)	ST	2

TABLE VII

THE PERFORMANCE OF USER-IOT INTERACTION DETECTION.

Device	User-IoT Interaction	Precision	Recall	F-Score
EP	Turn on/off power	1.00	0.98	0.99
MC	Turn on/off moving detection	0.5	1.00	0.67
	Moving detection	1.00	0.95	0.97
	User connection	1.00	1.00	1.00
	Sleep mode change	0.69	0.98	0.81

Answer to RQ2: We can accurately estimate the number of IoT devices of the same type behind NAT with the average error less than 5%.



(a) Amazon Echo

(b) Mijia Plug

Fig. 8. Scatter plot of estimated/actual numbers for IoT devices behind NAT.

3) User-IoT Interaction Detection: We repeat a certain user-IoT interaction multiple times to collect traffic samples. The samples are divided into training samples and testing samples for generating SPs and calculating recall, respectively. To compute precision, we introduce two additional sets of traffic samples exclusive of the target user-IoT interaction, namely, traffic samples of other user-IoT interactions and traffic samples collected in the idle state. Our experiments involve two IoT devices: Ezviz Plug (EP) and Mijia Camera (MC). For each device, we manually trigger all its user-IoT interactions except those that do not generate traffic or are difficult to repeat (e.g., firmware updating). Each interaction is repeated 50 times to collect training and testing samples with a ratio of 1:4.

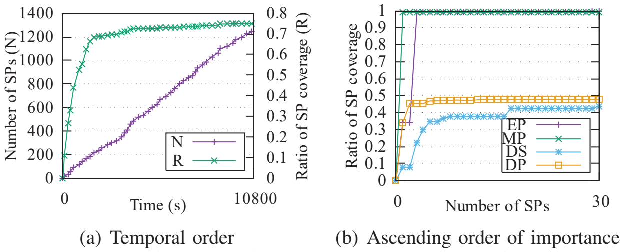
Table VII lists the user-IoT interactions, along with detection precision, recall, and F-Score. We see that the results are promising since we can identify most interactions with a high F-score. In particular, the "Turn on/off power" interaction of EP can be accurately identified. However, the traffic of "Turn on power" is the same as that of "Turn off power". Admittedly, we cannot distinguish between them without inspecting the application-layer content, but at least we can identify the power on-off state transition of EP. We face the same situation when detecting "Turn on/off moving detection" of MC, but accurately detecting other interactions like "Moving detection" indicates that moving detection has already been turned on.

C. Scalability Analysis in Practical Deployment

As we train CNN models separately for each device beforehand, the scalability of our system mainly depends on the detection stage. The detection includes two computation-critical tasks, namely, CNN model processing and SP matching. CNN model processing does not consume too many computational resources, especially when the number of model parameters is small (less than 20MB of memory usage under our settings).

The main computational cost is attributed to SP matching. The time complexity of SP matching is $O(pq)$, where p is the number of packets, and q is the length of an SP. Since q is a constant in detection, the time complexity of our system is almost linear to the number of packets. Therefore, our system is competent at large-scale deployment.

During training, there is no need to observe the traffic of an IoT device for a very long period of time so to generate SPs. Thus, the training time, along with the needed size of IoT traffic, is limited. As shown in Fig. 9(a), we generate SPs in different time periods for Amazon Echo. As time proceeds, the number of SPs becomes larger. If we use the SPs to match all the original packets, we derive the ratio of SP coverage to represent the proportion of the original packets that can be characterized by these SPs. We see that the number of SPs increases smoothly and (almost) constantly, while the ratio of SP coverage grows very slowly after an initial rapid climb. That is, the marginal utility of generating SPs to characterize the original packets is drastically reduced after the initial period (e.g., 30 minutes or one hour). As shown in Fig. 9(b), the SPs of the x-axis are arranged in ascending orders of their importance defined in (6). Mijia Plug (MP) has only 9 different SPs, and the first SP characterizes 95% of the original packets. DuSmart Speaker (DS) induces more than 1,900 different SPs in one hour, and 10 of them characterize over 35% of the original packets. Although 35% is not high, using only 10 SPs still achieves excellent detection performance.



(a) Temporal order (b) Ascending order of importance

Fig. 9. The trend of the ratio of SP coverage as the number of SPs increases in temporal order and in ascending order of their importance.

When our system is deployed in large-scale networks, training CNN models for each device separately is favorable. For instance, no more than 5 SPs are sufficient to represent Ezviz Plug's behavior without undermining detection performance. The packet time intervals of PIX-STAR Photo-frame exceeds 360 seconds, and one needs to increase the CNN time window. All these customized operations reduce computational overhead while assuring or even improving detection performance.

V. Related Work

With the prosperity of IoT, traffic fingerprinting is gradually leveraged for detecting IoT devices. For example, Amar et al. disclosed the feasibility of detecting IoT devices through traffic fingerprinting by case studies [15]. They did not build a detection system. Yang et al. implemented an IoT discovery system through actively probing the IPv4 space [16]. However, many devices residing behind NAT limit the application scope of their system. Acar et al. revealed that, when a user behind NAT occasionally accesses a phishing website with DNS rebinding scripts, IoT devices behind the same NAT can be fingerprinted [17]. Such vulnerability-based methods are not general and are unlikely to be adopted in network management.

Several studies built machine learning based passive traffic fingerprinting systems for detecting IoT devices. According to the learning algorithms, they can be categorized into feature-based and deep learning-based studies. Feature-based studies craft traffic feature vectors, and then employ supervised classification algorithms to conduct training and testing [18]–[21]. Deep learning-based studies use raw data as input. Then, CNN and Recurrent Neural Network (RNN) are used to automatically generate features and make the classification [22], [23]. These studies have confirmed that passive traffic fingerprinting empowers the network manager to accurately identify IoT devices [5], [18]–[20], [22]–[24]. For example, Bezawada et al. extracted 20 features, including entropy and header features, for each packet and obtained consistently good results [19]. Meidan et al. designed a session based classifier [20]. Jafari et al. collected physical layer information from several ZigBee devices and achieved detection with neural networks [22].

Our work is fundamentally different. Existing studies, such as [5] and [15], have an underlying assumption: each observed IP address just hosts a single (IoT) device. Given a period of traffic traces associated with an IP address, their model categorizes the traces into only one class (i.e., one IoT device). However, it is pretty common for IoT devices to be deployed behind NAT, generating complicated traffic sharing the same IP address. Naturally, a period of traffic traces associated with an IP address should be categorized into one or more classes. The underlying assumption makes existing studies practical only in limited scenarios. Additionally, existing studies may require the training/testing traffic traces to be split into separate samples, hence not well suited to the continuously arriving traffic.

Eliminating the underlying assumption and prerequisite, we aim to detect IoT devices from *an ISP's perspective*. When designing our system, we consider the common fact that many IoT devices are hidden behind NAT and the traffic's continuous arriving property. Besides just detecting the presence of IoT devices, we can accurately estimate their numbers, and detect user-IoT interactions, in an online fashion without the need to explicitly split traffic traces. Extensive experiments proved the effectiveness of our system. Although Thangavelu et al. developed a distributed device fingerprinting technique (DEFT) from an ISP's perspective [25], DEFT requires control over routers, and necessitates extensive router configurations. Therefore, its scalability and immediate availability are limited.

VI. Conclusion

Motivated by the fact that many IoT devices are placed behind NAT and hidden to network administrators, we make the first effort towards passively pinpointing hidden IoT devices from an ISP perspective. Our system can accurately identify IoT devices, estimate their numbers, and detect user-IoT interactions via spatial-temporal traffic fingerprinting, even when devices are hidden behind NAT. Extensive evaluation showed that our system can generally identify IoT devices with an F-Score above 0.999, and estimate the number of the same type of IoT device behind NAT with an average error below 5%. The system can scale up to large networks and work in an online fashion (identify IoT devices in just a few minutes after they are connected) because of its indispensability to user intervention during training and low time complexity during detection. In the future, we will further explore our system's potential in detecting user-IoT interactions.

Acknowledgement

This work was supported in part by National Natural Science Foundation (61972313, U1736205, U1766215, 61822205), National Key R&D Project (2016YFB0901900), Postdoctoral Science Foundation (2019M663725), CCF-NSFOCUS KunPeng Research Fund, of China. The work by John C.S. Lui was supported in part by the GRF 14201819.

REFERENCES

[1] "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)." https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2019.

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, 2017.

[3] A. O. Prokofiev and Y. S. Smirnova, "Counteraction against Internet of Things botnets in private networks," in *Proc. IEEE EIConRus*, 2019.

[4] S. Soltan, P. Mittal, and H. V. Poor, "BlackIoT: IoT botnet of high wattage devices can disrupt the power grid," in *Proc. USENIX Security*, 2018.

[5] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, 2018.

[6] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, 1977.

[7] C. Bepery, S. Abdullah-Al-Mamun, and M. S. Rahman, "Computing a longest common subsequence for multiple sequences," in *Proc. IEEE EICT*, 2015.

[8] K. Chen, Z. Zhang, J. Long, and H. Zhang, "Turning from TF-IDF to TF-IGM for term weighting in text classification," *Expert Systems with Applications*, 2016.

[9] S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-time patient-specific ECG classification by 1-D convolutional neural networks," *IEEE Transactions on Biomedical Engineering*, 2016.

[10] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-time motor fault detection by 1-D convolutional neural networks," *IEEE Transactions on Industrial Electronics*, 2016.

[11] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2014.

[12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, 2014.

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[15] Y. Amar, H. Haddadi, R. Mortier, A. Brown, J. Colley, and A. Crabtree, "An analysis of home IoT network traffic and behaviour," *arXiv preprint arXiv:1803.05368*, 2018.

[16] K. Yang, Q. Li, and L. Sun, "Towards automatic fingerprinting of IoT devices in the cyberspace," *Computer Networks*, 2019.

[17] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based attacks to discover and control local IoT devices," in *Proc. ACM IoT S&P*, 2018.

[18] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker, "Passive data link layer 802.11 wireless device driver fingerprinting." in *Proc. UNISEX Security*, 2006.

[19] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "Iotsense: Behavioral fingerprinting of IoT devices," *arXiv preprint arXiv:1804.03852*, 2018.

[20] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis," in *Proc. ACM SAC*, 2017.

[21] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "IoT Sentinel: Automated device-type identification for security enforcement in IoT," in *Proc. IEEE ICDCS*, 2017.

[22] H. Jafari, O. Omotere, D. Adesina, H.-H. Wu, and L. Qian, "IoT Devices Fingerprinting Using Deep Learning," in *Proc. IEEE MILCOM*, 2018.

[23] S. Aneja, N. Aneja, and M. S. Islam, "IoT Device Fingerprint using Deep Learning," in *Proc. IEEE IOTAIS*, 2018.

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

[25] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy, "DEFT: A Distributed IoT Fingerprinting Technique," *IEEE Internet of Things Journal*, 2018.