Inferring Hidden IoT Devices and User Interactions via Spatial-Temporal Traffic Fingerprinting

Xiaobo Ma, *Member, IEEE*, Jian Qu, Jianfeng Li, John C. S. Lui, *Fellow, IEEE*, *ACM*, Zhenhua Li, *Senior Member, ACM*, Wenmao Liu, and Xiaohong Guan, *Fellow, IEEE*

***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. Additionally, inferring fine-grained user interactions of IoT devices is also an interesting yet unresolved problem. In this paper, we design an efficient and scalable system via spatial-temporal traffic fingerprinting from an ISP's perspective in consideration of practical issues like learning-testing asymmetry. Our system can accurately identify typical IoT devices in a network, with the additional capability of identifying what devices are hidden behind NAT and the number of each type of device that share the same IP address. Our system can also detect user interactions and meanwhile identify their (concurrent) number through a multi-output regression model. Through extensive evaluation, we demonstrate that the system can generally identify IoT devices with an F1-Score above 0.999, and estimate the number of the same type of IoT device behind NAT with an average error below 5%. By studying 29 user interactions of 7 devices, we show that our system is promising in detecting user interactions.**

***Index Terms*— IoT traffic analysis, IoT device detection, network monitoring.**

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 via these vulnerabilities [5]–[8].

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. If ISPs can recognize hidden IoT devices, the primary security policy that they can use is to examine the presence of public vulnerabilities, and in turn take defensive measures in advance. For example, if a hidden device like Bosch IP camera, which has a high-risk vulnerability of CVE-2021-23853 [9], is detected, ISPs can filter out packets with abnormal HTTP headers to prevent malicious code injection. In addition, ISPs can notify IoT users of the vulnerabilities and urge them to fix the vulnerabilities (e.g., updating the firmware) by official announcements.

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.

Manuscript received March 3, 2021; revised August 15, 2021; accepted September 10, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor T. He. Date of publication September 27, 2021; date of current version February 17, 2022. This work was supported in part by the National Natural Science Foundation under Grant 61972313 and Grant 61822205, in part by the Postdoctoral Science Foundation under Grant 2019M663725 and Grant 2021T140543, in part by the Fundamental Research Funds for the Central Universities, and in part by China Computer Federation (CCF)-NSFOCUS KunPeng Research Fund of China. The work of Xiaobo Ma was supported by Cyrus Tang Foundation. The work of John C. S. Lui was supported in part by the Research Grants Council (RGC) under Grant R4032-18. *(Corresponding author: Xiaobo Ma.)*

Xiaobo Ma, Jian Qu, and Xiaohong Guan are with the MOE Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an 710049, China, and also with the Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: xma.cs@xjtu.edu.cn; qj904154277@stu.xjtu.edu.cn; xhguan@xjtu.edu.cn).

Jianfeng Li is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: jfli.xjtu@gmail.com).

John C. S. Lui is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: cslui@cse.cuhk.edu.hk).

Zhenhua Li is with the School of Software, Tsinghua University, Beijing 100084, China (e-mail: lizhenhua1983@tsinghua.edu.cn).

Wenmao Liu is with NSFOCUS Inc., Beijing 100089, China (e-mail: liuwenmao@nsfocus.com).

Digital Object Identifier 10.1109/TNET.2021.3112480

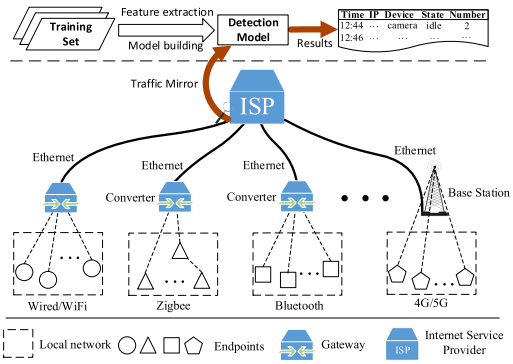


Fig. 1. An ISP's perspective of passively inferring hidden detecting IoT devices and user interactions.

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 system detects IoT devices from an ISP's perspective. In other words, it can be used to detect all IoT devices that have access to the Internet. Moreover, our system is deployed on the border of a monitored network (e.g., a stub network, a city-scale network) to inspect the inbound-outbound TCP/IP packets. The IoT devices in the monitored network will be identified by our system as their traffic traverses the border. Although the IoT devices may use various wireless technologies, either short ranges (e.g., Zigbee, Bluetooth) or long ranges (e.g., 4G/5G, LoRa), to get connected to the monitored network, these wireless technologies are transparent to our system. That is, whatever wireless technologies an IoT device uses, our system could detect IoT devices at the upstream network border, since it is deployed from an ISP's perspective on the border of the monitored network, where the traffic of all devices are converted into TCP/IP packets.

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?

Tackling the above two research questions enables ISPs to be aware of "which and how many IoT devices reside in a network" for security maintenance. Further inferring fine-grained user interactions of IoT devices (i.e., "which user interactions are posed to IoT devices") is also an interesting yet unresolved problem. A technique addressing this problem is important for two reasons. First, a security critical company that requires systematic evaluation of its deployed IoT devices would rely on such a technique to measure how well its devices conform to a set of established user interaction criteria. Second, both IoT users and manufacturers would be concerned about whether the usage of certain IoT devices introduces new privacy concerns, since ISPs could potentially analyze network traffic to infer user interactions of IoT devices.

We, therefore, endeavor to further answer the following RQ.

RQ3. If the answers to RQ1 and RQ2 are positive, can we further detect the user interactions of 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 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 networks (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 infer hidden IoT devices and user interactions from an ISP's 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.
- To detect user interactions of IoT devices, we propose a multi-output regression model (instead of a classification model) that can not only detect multiple concurrent interactions but also identify the number of each interaction in the active-state traffic.
- Through extensive evaluation in a network with nearly 3,000 users, we demonstrate that our system can generally detect IoT devices with an F1-Score above 0.999, and estimate the number of the same type of IoT device behind NAT with an average error below 5%.

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
Hemu SmartPlug (HP)	Plug	Aixiangji	2018
HUAWEI AI Speaker Mini (HS)	Speaker	HUAWEI	2018
HUAWEI desk lamp (HL)	Light	Gubei	2018
ORVIBO light strip (OL)	Light	Espressif	2018
Iateey water bar (IW)	Kettle	AI-Link	2019
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

We collect traffic of user interactions on 7 devices and conducted experiments to detect 29 user interactions. The results show that our method is promising in classifying user interactions and detecting concurrent interactions. We reveal that the performance may vary from device to device, depending on the interaction's stability and distinguishability in terms of traffic characteristics.

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

II. UNDERSTANDING IOT DEVICES AND THEIR TRAFFIC CHARACTERISTICS

To understand the behavior of IoT devices, we collect and analyze the traffic traces of 22 typical IoT devices. The traffic traces come from two sources. One is via our testbed where we capture the traffic traces of 13 IoT devices. The other is a public dataset published by the University of New South Wales [10], offering us the traffic traces of 9 additional IoT devices. We detail these IoT devices' names, functions, manufacturers, and year of manufacture (YoM) in Table I. Note that all IoT devices under our 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 *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

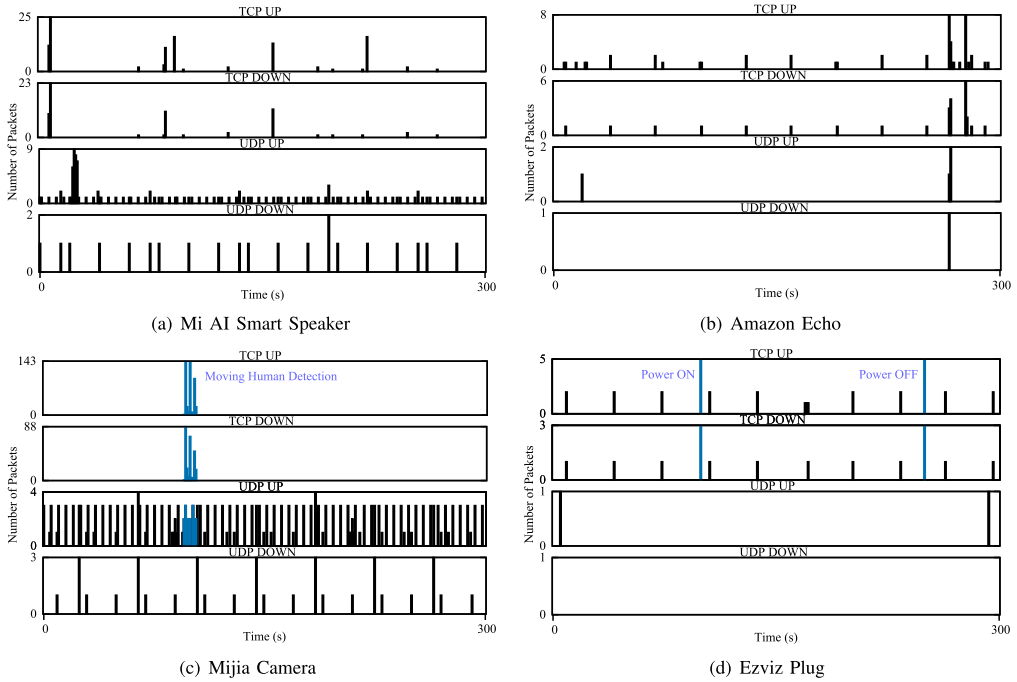
As different tasks are executed, an IoT device transits from one state to another. Note that the initialization state is transient, the idle state is the default state, and the active state only appears when a device is executing tasks. Therefore, one can expect that 1) the idle-state traffic is persistent; 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.

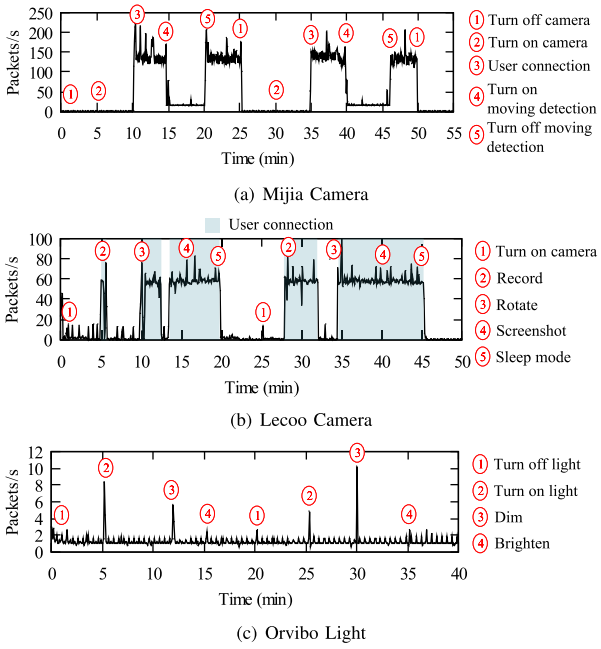
To further observe user interactions of IoT devices, we collected the traffic of three typical IoT devices, as shown in Fig. 3. It can be found that the packet transmission rate varies as different user interactions occur. Take Mijia Camera as an example in Fig. 3(a). The traffic was captured in 55 minutes, involving 5 user interactions. At the beginning, we turned off the camera through the mobile app, and then turned on the camera. It can be observed that, even when the camera is turned off, the packet transmission remains. This



(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).



(a) Mijia Camera

(b) Lecoo Camera

(c) Orvibo Light

Fig. 3. Different user interactions of three typical IoT devices. A significant increase in traffic can be observed when user interactions are triggered.

means that the "turn off" operation of Mijia Camera can be regarded as "sleep mood" (rather than "power off"). At the 10th minute, we connected to the camera to remotely see the live video stream through the mobile app, resulting in rapid increase of packet transmission. After that, we modified some configurations about moving detection through the mobile app and stayed at the configuration user interface (from the 15th to the 20th minute). The traffic transmission rate during this period drops to a lower range but still greater than that at the beginning.

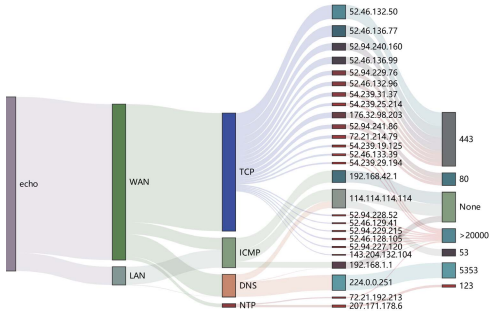


Fig. 4. Sankey diagram of Amazon echo traffic. Bars from left to right represent device name, WAN or LAN, protocol, server IP address, and port. "None" means that the protocol ICMP has no port.

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. 4 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.

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

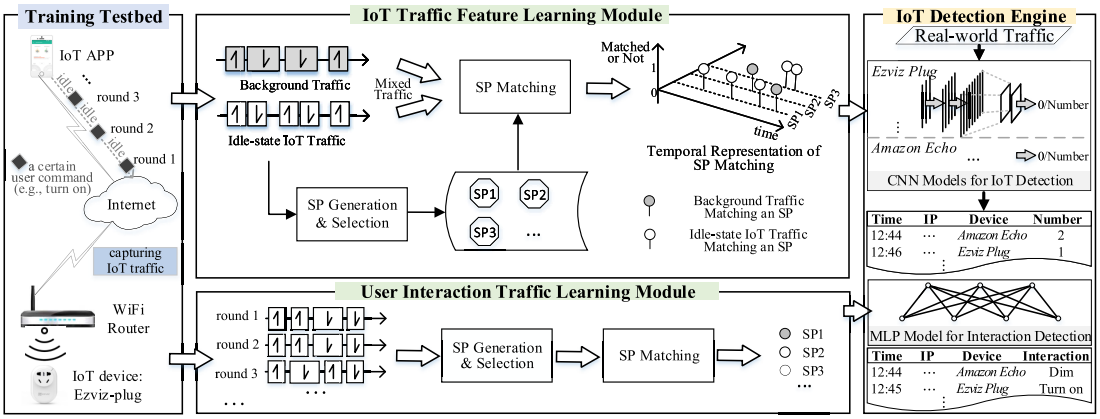


Fig. 5. The system architecture of detecting IoT devices and user interactions (SP: Sequence Profile).

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. 5. In the training stage, we capture and label traffic for each IoT device. To detect the presence of IoT devices, we keep IoT devices in idle state without user intervention. 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 the detection stage, the IoT detection engine identifies IoT devices and estimates their numbers in real-world traffic. To detect the user interactions, we manually label the different user interactions of devices. The user interaction traffic learning module generates several features to represent the interactions for each device. In the detection stage, the IoT detection engine detects different user interactions whose features have been previously learned. 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

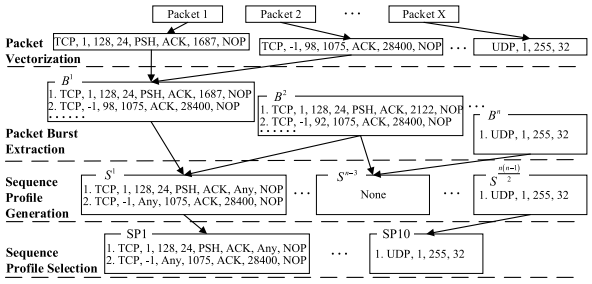


Fig. 6. An example of generating SPs via hierarchical feature extraction.

TABLE III
TRAINING PARAMETER SETTINGS

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.

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. 6.

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. We would like to point out that features like domain names and IP addresses of IoT-contacting servers are not used, since they are not reliable due to CDNs, and may not be discriminative when different IoT devices use the same cloud services.

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. 6, 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 [11], [12]. 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

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

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 [13], 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 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". Compared with TF-IDF, our importance-based method focuses on frequent common sequences, avoiding the scarcity of SPs and supporting a small detection time window. 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. 5, 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. 8 shows examples of SP matching over DuSmart Speaker's traffic and background traffic, which will be explained in detail in Sec. IV-A.

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 [14]–[16]. 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 [17], [18]. 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 Interaction Detection

If we already know the presence of an IoT device, say A, behind a certain IP address, we can train a new model to identify specific user interactions of device A. Intuitively, each interaction can be seen as a composite of one or more SPs. Therefore, we extract SPs for each interaction and select the SPs that represent it. Unlike the previous CNN model for IoT detection based on idle-state traffic, we do not map SPs to the time axis, since the traffic of an interaction generally happens at discrete time points or in a short time interval (i.e., a burst). Instead, we directly employ an SP-matching feature vector, each element of which is the corresponding SP's number of matches against traffic under inspection, as detection features of user interactions.

Detecting user interactions can be regarded as a standard traffic classification task, either a bi-classification one or a multi-classification one. Should the task be a bi-classification one, we need to train a bi-classifier for each individual interaction of a device. The bi-classifier will determine whether the corresponding interaction exists in the traffic. Should the task be a multi-classification one, we only need to train one multi-classifier for a device. The multi-classifier will figure out which interaction occurs in the traffic from among a set of interactions. The multi-classifier reduces training overhead, while it may fail if two different interactions of a device occur in a short period of time.

For a certain IoT device, the interactions are normally sparsely distributed over time. Therefore, in a small time window, the interactions would be rare. For example, there are five devices of the same type sharing one same IP address using NAT. This device has three different types of user interactions. Assume that, for each device, each type of interactions is triggered 20 times (very frequently) a day and can be modelled as Poisson process. Then, within a 6-second time window, the probability that an interaction only occurs once exceeds 95%.

To consider the special cases where multiple user interactions occurring in a short period of time, we propose a multi-output regression model (instead of a classification model) that can not only detect multiple interactions but also identify the number of each interaction. To train this model, besides feeding training samples involving only one interaction, we also synthesize training samples that are a mixture of different or identical interactions in a short period of time to simulate the aforementioned special cases. We use multi-layer perceptron (MLP) as the regression model, since it

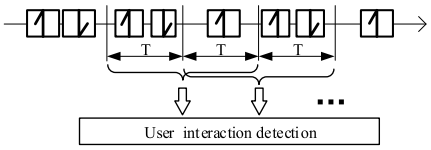


Fig. 7. User interaction detection using the sliding window. After finishing training the detection model, the traffic within the time window of $2T$ with a sliding step of T is inspected for detecting user interactions.

has multiple outputs in one model and good nonlinear fitting ability.

Recall that detecting user interactions depends on training a model using the active-state traffic. Compared to inspecting idle-state traffic without user intervention, collecting active-state traffic is labor-intensive because one needs to repeat the same type of interaction many times, especially those interactions relying on physical contact with IoT devices. This indicates that a detection model without the need to collecting a large number of training samples is more favorable than a sample-intensive model like deep neural networks.

While deploying the user interaction detection model to inspect real-world traffic, we can use the sliding window mechanism, as shown in Fig. 7. Suppose that the traffic generated by each user interaction of a device does not exceed T seconds. To ensure that each interaction is inspected within at least one time window, we set the width and the sliding step of the sliding window to be $2T$ and T, respectively.

IV. Evaluation

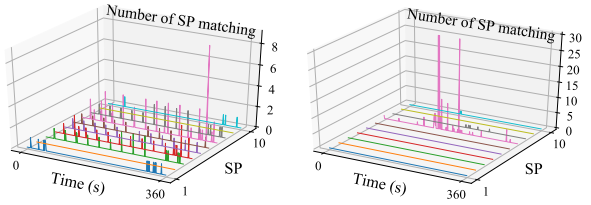
We evaluate the performance of our system in IoT device identification, population estimation, user interaction detection, and analyze its scalability in practical deployment.

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 [10]. 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



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

Fig. 8. Time arrays of matching SPs against positive/negative traffic samples.

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

(i.e., CNN Time Window in Table V). These time arrays can be directly fed into CNN models for IoT training and detection.

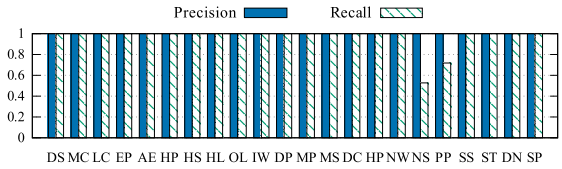
Fig. 8 exemplifies time arrays of matching SPs of DuSmart Speaker against positive and negative traffic samples. We see that time arrays for positive samples are regular with occasional irregularity, while those for negative samples are abrupt and irregular. In this example, for ease of demonstration, we build positive samples using pure idle-state traffic of DuSmart Speaker, and negative samples using background traffic.

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 the traffic of all other devices.

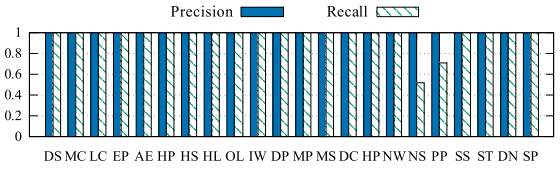
Tables IV and V list our parameter settings. Table IV is for calculating the distance between two PVs in (1), where Weight and Max mean w_k and m_k, respectively. Table V includes parameters including maximum SP number, distance threshold d, interval of bursts, and training parameters.

B. Results and Insights

1) IoT Device Detection: Fig. 9 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. 9(a) is the results using our default features without the information of domain name, IP address and port, while Fig. 9(b) shows the results after adding such information into our system. We observe that the performance is not



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



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

Fig. 9. 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	HP	1
HS	1	HL	1	OL	1
IW	1	DP	1	MP	1
MS	2	DC	1	HP	2
NW	1	NS	13	PP	12
SS	1	ST	2	DN(SP)	2(4)

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.

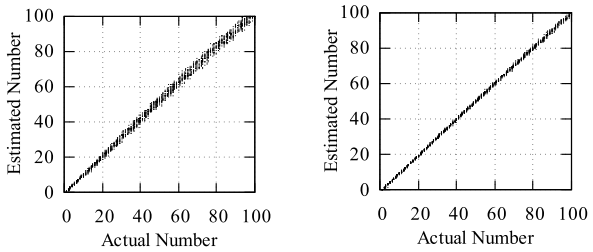
Both precision and recall increase as the detection time proceeds. Further, we want to know the minimum time for successfully detecting IoT devices. We test the detection response time for all devices (i.e., the time lag between an IoT device being connected and the successful detection) with the F1-Score 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 F1-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. 10 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.

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

3) User Interaction Detection: We repeat a certain user interaction multiple times to collect traffic samples. Our experiments involve seven IoT devices. Four of them are fro1m our testbed: Mijia Camera, Lecoo Camera, Orvibo Light and HUAWEI AI Speaker Mini. The others are from the Mon(IoT)r dataset [19]. The Mon(IoT)r dataset contains user-IoT interaction traces from 55 distinct IoT devices. However, most of them only have two interactions (e.g., on and off), and many interactions are triggered for a very small number of times. Consequently, we select three devices, namely, Amazon Echo Plus, TP-LINK Bulb and Honeywell Thermostat. The selected devices involve no less than three interactions, each of which is triggered at least 30 times, thereby ensuring sufficient training samples. For each device from our testbed, we manually trigger all its interactions except those that do not generate traffic or are difficult to repeat (e.g., firmware updating). Each interaction is repeated 50 times.

We perform two experiments for evaluating the detection performance in normal cases and special cases, respectively. The first experiment evaluates the detection model's capability of distinguishing different interactions of a target device, focusing on the normal cases where only one interaction of the target device exits in the traffic within the sliding window under inspection. In the second experiment, we are interested in the detection model's performance in special cases where multiple interactions of the target device exist in the traffic within the sliding window under inspection. When performing both experiments for a target device, in the background traffic, we also consider the practical situation of the presence of multiple devices of the same type as the target device, as well as other types of devices, behind an IP address.

In the first experiment, Support Vector Machine (SVM) is used as the classifier. For each type of IoT device, we train a classifier to detect its interactions. To calculate the F1-Score of the classifier, we use 10-fold cross validation. The classification results will be accumulated across all rounds of traffic samples. Table VII lists the detection performance of different interactions of all devices. "Prediction Results" describes the distribution of model prediction results when the traffic sample is known to be a certain type (i.e., different interactions, or background traffic), in the form of a sparse representation of

TABLE VII
EXPERIMENT RESULTS OF DETECTING USER-IOT INTERACTIONS

Device	User Interaction	Prediction Results	F1-Score	RTE-1	RTE-2
Mijia Camera	① Turn on/off camera	① 98% ③ 2%	0.99	0.08	0.02
(50 samples)	② User connection	② 98% ③ 2%	0.99	0.14	0.07
	③ Turn on/off moving detection	③ 100%	0.99	0.06	0.05
	ⓑ Background traffic	ⓑ 100%	0.99	-	-
Lecoo Camera	① Turn on camera	① 84% ② 2% ③ 14%	0.87	0.49	0.45
(50 samples)	② Record	② 40% ⓑ 60%	0.56	0.73	0.70
	③ Rotate	① 8% ③ 92%	0.89	0.38	0.37
	④ Screenshot	④ 100%	1.00	0.08	0.07
	⑤ Sleep mood	⑤ 100%	1.00	0.14	0.14
	ⓑ Background traffic	ⓑ 100%	0.77	-	-
Orvibo Light	① Brighten	① 40% ② 54% ④ 6%	0.46	0.67	0.71
(50 samples, separate interactions)	② Dim	① 32% ② 66% ③ 2%	0.59	0.67	0.64
	③ Turn off light	③ 56% ④ 44%	0.53	0.72	0.67
	④ Turn on light	① 12% ② 4% ③ 52% ④ 42%	0.44	0.64	0.62
	ⓑ Background traffic	ⓑ 100%	1.00	-	-
Orvibo Light	① Brighten/Dim	① 96% ② 4%	0.96	0.20	0.10
(100 samples, merged interactions)	② Turn on/off power	① 3% ② 97%	0.97	0.25	0.16
	ⓑ Background traffic	ⓑ 100%	0.99	-	-
HUAWEI AI Speaker Mini	① Play the next song	① 74% ③ 22% ⑥ 4%	0.82	0.48	0.40
(50 samples)	② Play the previous song	② 100%	1.00	0.74	0.66
	③ Play music	① 6% ③ 94%	0.87	0.56	0.52
	④ Stop music	④ 100%	0.99	0.42	0.38
	⑤ Volume down	④ 2% ⑤ 70% ⑥ 28%	0.64	0.71	0.72
	⑥ Volume up	⑤ 50% ⑥ 50%	0.55	0.73	0.76
	ⓑ Background traffic	ⓑ 100%	1.00	-	-
Amazon Echo Plus	① Change color	① 42% ② 30% ③ 18% ④ 10%	0.42	0.78	0.74
(40 samples)	② Dim	① 45% ② 27% ③ 20% ④ 8%	0.31	0.79	0.83
	③ Turn off Echo	① 12% ② 8% ③ 55% ④ 25%	0.48	0.77	0.74
	④ Turn on Echo	① 2% ② 15% ③ 38% ④ 45%	0.47	0.80	0.81
	ⓑ Background traffic	④ 2% ⓑ 98%	0.99	-	-
TP-LINK Bulb	① Change color	① 100%	1.00	0.13	0.10
(40 samples)	② Dim	② 100%	1.00	0.15	0.07
	③ Turn off power	③ 100%	1.00	0.16	0.05
	④ Turn on power	④ 100%	1.00	0.19	0.13
	ⓑ Background traffic	ⓑ 100%	1.00	-	-
Honeywell Thermostat	① Turn off thermostat	① 53% ② 44% ③ 3%	0.63	0.60	0.61
(36 samples)	② Turn on thermostat	① 11% ② 86% ③ 3%	0.75	0.62	0.62
	③ Set thermostat	① 3% ③ 97%	0.92	0.80	0.78
	ⓑ Background traffic	③ 8% ⓑ 92%	0.96	-	-

confusion matrix. For example, "①98% ③2%" means "Sleep mode change" of Mijia Camera are correctly identified in 98% cases while falsely detected as "Turn on moving detection" in the remaining 2% cases. "F1-Score" will be derived from the confusion matrix.

In the second experiment, the MLP regression model is used to predict the number of each interaction when multiple interactions occur simultaneously. In the training process, the training samples are randomly mixed to simulate concurrent interactions. In our experiment, we set the maximum number of concurrent interactions to be 10. This number is tunable. If more than 10 interactions concurrently occur (though this rarely happens), one can just increase this parameter and retrain the model. During the testing process, we increase the number of concurrent interactions from 0 to 10 with a step of one. For a certain number of concurrent interactions, we select the test samples uniformly at random from all possible interactions for 100 times, resulting in 100 samples. Again, we use 10-fold cross validation to evaluate the MLP regression model. The performance metric Relative Total Error (RTE) is defined as:

$$\text{RTE} = \frac{\sum_{n=1}^{N} |X_n - [Y_n]|}{\sum_{n=1}^{N} |X_n|}, \tag{8}$$

where N represents the number of test samples, X_n and Y_n are the actual number of interactions and the predicted number of interactions of the nth test sample, respectively, and $[Y_n]$ represents the rounding of Y_n.

In Table VII, RTE-1 represents the RTE of the MLP regression model. We see that, when different interactions of the same type of device are mixed, the error varies broadly

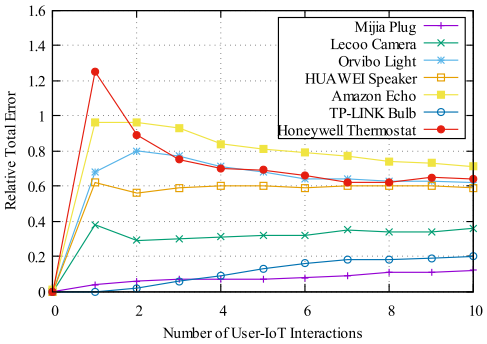


Fig. 11. Relationship between RTE and the number of concurrent interactions.

ranging from 0.06 to 0.92. To further improve the performance of detecting concurrent interactions, we introduce stream SPs as additional features. A stream SP is an SP that requires all elements of the SP belonging to the same four tuple stream: source IP address, destination IP address, source port, and destination port. These additional features enhance the distinguishability of mixed traffic streams from different devices, thereby beneficial to detecting concurrent interactions. RTE-2 lists the errors that use additional 10 stream SPs for detecting each interaction. In most cases, it outperforms RTE-1.

To gain insight into the detection sensitivity to the number of concurrent interactions, we trained the models with 7 different devices, and tested the RTE under the different number of concurrent interactions. The results are shown in Fig. 11. Note that the denominator of the RTE cannot be 0, and in this case, we set it to the sample number. For those devices with low errors like Mijia Plug and TP-LINK Bulb, there is a trend that the RTE increases as the number of concurrent interactions increases. For other devices, no such trend exists. The reason for the high error is generally due to the significant variation of the interaction's traffic characteristics. For example, in 50% of cases, Honeywell Themostat's interaction "set thermostat" does not trigger any traffic, while in the left 50% of cases, it triggers traffic with diverse characteristics.

Overall, our model has good interaction classification ability on most devices like Mijia Camera and TP-LINK Bulb. Note that, although RTE evaluates the effect of detecting concurrent interactions, and F1-Score represents the effect of classifying individual interactions, there is a strong correlation between RTE and F1-Score. Specifically, if F1-Score is close to 1, there is a high probability that RTE will be very low. This implies that accurately classifying individual interactions is a cornerstone of detecting concurrent interactions of a device.

C. Analysis of Distinguishability

We would like to point out that some interactions of a device may be indistinguishable. For example, by observing data packets with different interactions of Orvibo Light, we found that the packets of brighten and dim are indistinguishable, and turn on and off power are also indistinguishable. Such indistinguishability is also reflected in the "Prediction Results" in Table VII, where these pairs of similar interactions have high probabilities to be misclassified as each other.

To further verify whether the indistinguishable interactions of Orvibo Light, we conduct analysis from the perspective of data complexity. Specifically, we select four features that

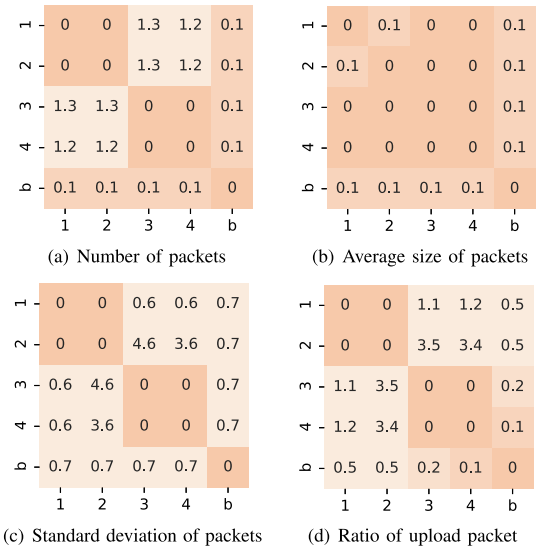


Fig. 12. Fisher's discriminant ratio matrix for four features. Each element in the matrix represents the Fisher's discriminant ratio of the Orvibo Light's interactions corresponding to the horizontal and vertical coordinates. Large values mean significant difference between the two interactions.

are usually used in traffic classification tasks, and calculate their Fisher's discriminant ratio (FDR) [20] for each pair of similar interactions, as shown in Figure 12. The higher FDR is, the easier it is to distinguish between the two interactions. We see that no features can clearly distinguish between interaction 1 and interaction 2, and interaction 3 and interaction 4.

As demonstrated in Table VII, if we merge similar (separate) interactions that are indistinguishable into one type, the performance metric would be improved. This implies that the classification performance metric also depends on the granularity of labeling interaction types.

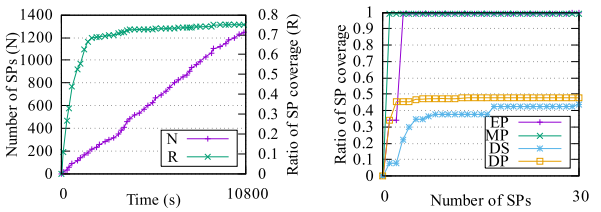
Answer to RQ3: Our method is promising in classifying user interactions and detecting concurrent interactions. Our method can detect user interactions with an average F1 score of 0.80, and half of the user interactions with F1 scores above 0.90.

D. Scalability 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. 13(a), we generate SPs



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

Fig. 13. The trend of the ratio of SP coverage as the number of SPs increases in temporal order and in ascending order of their importance.

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. 13(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.

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. Discussion

The key to passively detecting IoT devices and user interactions is training a detection engine that incorporates the traffic characteristics of various IoT devices. Our work provides a solution to training such a detection engine. However, labeling training samples at scale, especially those for user interactions, remains labor-intensive. Exhaustively labeling training samples is impossible given the increasing number and diversity of IoT devices. A possible way is to outsource the labeling task to IoT users. For example, Sundaresan *et al.* and Schmitt *et al.* proposed to deploy physical hardware devices for collecting traffic samples on home networks by users [21], [22].

However, deploying hardware devices would be costly. In contrast, Huang *et al.* developed and released IoT Inspector to outsource the labeling task. IoT Inspector is an open-source software that allows users to collect, observe, and label the traffic from smart home devices on their own home networks [23]. Such a software-based outsourcing approach well complements our work. Note that the outsourcing approach is perfectly suited to labeling IoT devices, since users' marking MAC or IP addresses in home networks as different types of IoT devices only requires their temporary workload.

Compared to collecting training samples of idle-state traffic of IoT devices, collecting training samples of user interactions is labor-intensive due to its requirement of user intervention. One can outsource the labeling of user interactions to IoT users. The participating users may upload their interaction records, along with the device types, to the outsourcing platform and get a certain reward. Alternatively, the labeling could be performed by reverse-engineering the mobile app of the IoT device, and automatically driving the device into different user interactions through running the app on a simulator. Also, these labeling approaches for user interactions are well suited to automating labeling idle-state traffic of IoT devices.

Note that we base our detection method on the de facto traffic characteristics of existing IoT devices. A recent study by Apthorpe *et al.* presented a traffic-shaping method capable of defending against passively inferring user activities at the cost of higher bandwidth overhead [24]. As the users' growing need for a high level of privacy, we believe that some IoT devices may adopt defenses against traffic analysis. However, given the already established fact of the proliferation and the huge number of IoT devices, it can be argued that it is extremely challenging, if not impossible, for such defenses to be adopted at scale in the foreseeable future. We conjecture that the reasons are two-fold. First, passive traffic analysis can hardly be perceived by IoT users. Hence, privacy breaches due to passive traffic analysis may not attract attention of IoT users. Second, compared to designing defenses against (sophisticated) passive traffic analysis, producing reliable, effective, and protocol-secure (e.g., adopting SSL/TLS) IoT devices is the top priority for the vast majority of IoT manufacturers at the present stage.

There are situations where the traffic of an IoT device may be changed. For example, a hacker compromises an IoT device, and then utilizes this device to attack other devices. In this example, the simplest and most efficient method for the hacker to launch the attack is to deploy the attack program on the device operating system without modifying its original network function. Otherwise, the device owner may find that the device cannot be used properly, thereby likely to recover the operating system and consequently render the attack program completely removed. Therefore, the attack would just induce additional (attack) traffic without changing the original (IoT) traffic. In such a case, our system will still work as expected.

In the extreme case that the hacker would like to fully reshape the traffic characteristics and evade our detection, he may need to modify either the client side (i.e., the device firmware) or the server side (e.g., the IoT cloud servers), or both sides. The client-side modification, though difficult, is possible. However, successful server-side modification would expose the hacker due to its wide-ranging influence on all devices. Therefore, the client-side modification would be the only feasible choice for the hacker. However, as the client-side modification scales up to a number of IoT devices, our system could have a chance to re-learn the modified traffic fingerprints when the modification is captured by honeynet. More importantly, continuous operation of our system could also be aware of the sudden disappearance of a device's previous traffic fingerprints, as well as the appearance of new traffic fingerprints. Last but not least, our system is based on passive traffic analysis. Hence, the hacker cannot perceive the its deployment, drastically making the hacker confused about when and where the detection evasion is needed.

In IPv6 environment, our system will still play an important role in two aspects. First, in addition to translating the address between IPv6 hosts, IPv6 NAT (e.g., NAT-PT) also helps to translate IPv4 addresses to IPv6 addresses (and vice versa) of network devices because of the IPv4/IPv6 coexistence [25].

Therefore, the emergence of IPv6, though offering a tremendous number of IP addresses, may not completely eliminate NAT in the foreseeable future. Second, even in the case that all devices have an IPV6 address without NAT, our system can detect IoT devices because all primitive features, except the TTL (at the IP layer), originate from the transport layer and the layers above. These features remain unchanged in IPv6.

As a matter of fact, the major obstacle to deploying our system in IPv6 lies in the problem of IP layer encryption, i.e., IPsec. If IPsec is enabled by a device, our system will no longer be applicable due to the disappearance of the primitive features. Fortunately, there are currently few IoT devices using IPv6 on the market, not to mention the use of IPsec. More importantly, although IPsec is a mandatory part of an IPv6 specification, its use is optional as described in RFC 6434 [26]. The reason is that special-purpose devices may support only very limited applications, and an application-specific security approach may be sufficient, and some devices may run on extremely constrained hardware where the full IPsec architecture is not justified.

While our system could detect IoT devices for security purposes, it may also introduce privacy concerns. In the long run, the IoT manufacturers should hide the traffic fingerprint of the devices for private networks (e.g., home networks). This would definitely prevent anyone tapping on the network from inferring running IoT devices and other privacy (e.g., whether a home is occupied). However, in this case, ISPs would fail to make security polices for these IoT devices. Therefore, a compromise solution is to hide the traffic fingerprint of user interactions while retaining the fingerprint only indicative of the presence of a device. When it comes to the IoT devices in enterprise and industrial networks, it would be more favorable to keep all the traffic fingerprints, since such networks involve less privacy and require more security management.

VI. 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 [27]. They did not build a detection system. Yang *et al.* implemented an IoT discovery system through actively probing the IPv4 space [28]. 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 [29]. 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 [30], [31]. 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 [32]–[35]. Deep learning-based studies use raw data as input. Then, CNN and Recurrent Neural Network (RNN) are used to automatically generate features and perform the classification [36], [37]. These studies confirmed that passive traffic fingerprinting empowers the network manager to accurately identify IoT devices [10], [32]–[34], [36]–[38]. For example, Bezawada *et al.* extracted 20 features for each packet and obtained consistently good results [33]. Meidan *et al.* designed a session-based classifier [34]. Jafari *et al.* collected physical layer information from several ZigBee devices and achieved detection with neural networks [36].

There are another branch of works studying IoT device fingerprinting through directly inspecting wireless protocols, such as Zigbee [39], Bluetooth [40], and LoRa [41]. Different from our study, the systems of these works are usually deployed at gateways (e.g., IoT hubs) directly connected to IoT devices, capturing the wireless traffic at the physical layer. Since the deployment location is very close to the devices, lower-layer features like MAC addresses could be used detect devices with standard (and product-specified) MAC addresses. However, these systems are physical-protocol specific and their detection scope is limited within IoT hubs.

In recent years, several studies have found that the user interaction traffic of IoT devices may leak user privacy [42], [43]. Ren *et al.* analyzed the encryption of IoT devices for revealing information exposure of 81 devices [19]. Trimananda *et al.* used a sequence of packet lengths to identify the interactions [44]. OConnor *et al.* generated features for each TCP connection and designed an interaction classifier [45].

Our work is fundamentally different. Existing studies, such as [10] and [27], 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 infer IoT devices and user interactions 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. Additionally, in our design, we remove simple features that could be easily adapted (hence less reliable), including domain names, IP addresses and port numbers, yet without detection performance degradation. Besides just detecting the presence of IoT devices, we can accurately estimate their numbers, and detect user 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 [46], DEFT requires control over routers, and necessitates extensive router configurations. Therefore, its scalability and immediate availability are limited.

VII. 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 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 F1-Score above 0.999, and estimate the number of the same type of IoT device behind NAT with an average error below 5%. We performed experiments to show that our system is promising in detecting fine-grained user interactions. We revealed that the performance may vary from device to device, depending on the interaction's stability and distinguishability in terms of traffic characteristics. 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.

REFERENCES

[1] (2019). *Internet of Things (IoT) Connected Devices Installed Base Worldwide From 2015 to 2025 (in Billions)*. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[3] A. O. Prokofiev and Y. S. Smirnova, "Counteraction against Internet of Things botnets in private networks," in *Proc. IEEE Conf. Russian Young Researchers Electr. Electron. Eng. (EIConRus)*, Jan. 2019, pp. 301–305.

[4] S. Soltan, P. Mittal, and H. V. Poor, "BlackIoT: IoT botnet of high wattage devices can disrupt the power grid," in *Proc. USENIX Secur.*, 2018, pp. 15–32.

[5] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "IoT security: Ongoing challenges and research opportunities," in *Proc. IEEE 7th Int. Conf. Service-Oriented Comput. Appl.*, Nov. 2014, pp. 230–234.

[6] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, "IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?" *IEEE Signal Process. Mag.*, vol. 35, no. 5, pp. 41–49, Sep. 2018.

[7] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on IoT security: Application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[8] F. Hussain, R. Hussain, S. A. Hassan, and E. Hossain, "Machine learning in IoT security: Current solutions and future challenges," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1686–1721, 3rd Quart., 2020.

[9] (2021). *CVE-2021-23853 Detail*. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-23853

[10] A. Sivanathan *et al.*, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1745–1759, Aug. 2019.

[11] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Commun. ACM*, vol. 20, no. 5, pp. 350–353, May 1977.

[12] C. Bepery, S. Abdullah-Al-Mamun, and M. S. Rahman, "Computing a longest common subsequence for multiple sequences," in *Proc. 2nd Int. Conf. Electr. Inf. Commun. Technol. (EICT)*, Dec. 2015, pp. 118–129.

[13] K. Chen, Z. Zhang, J. Long, and H. Zhang, "Turning from TF-IDF to TF-IGM for term weighting in text classification," *Expert Syst. Appl.*, vol. 66, pp. 245–260, Dec. 2016.

[14] S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-time patient-specific ECG classification by 1-D convolutional neural networks," *IEEE Trans. Biomed. Eng.*, vol. 63, no. 3, pp. 664–675, Mar. 2016.

[15] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-time motor fault detection by 1-D convolutional neural networks," *IEEE Trans. Ind. Electron.*, vol. 63, no. 11, pp. 7067–7075, Nov. 2016.

[16] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 22, no. 10, pp. 1533–1545, Jul. 2014.

[17] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, *arXiv:1207.0580*. [Online]. Available: https://arxiv.org/abs/1207.0580

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.

[19] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. IMC*, 2019, pp. 267–279.

[20] J. M. Sotoca, J. Sanchez, and R. A. Mollineda, "A review of data complexity measures and their applicability to pattern classification problems," *Actas III Taller Nacional Mineria Datos Aprendizaje*, vol. 2005, pp. 77–83, Jan. 2005.

[21] S. Sundaresan, S. Burnett, N. Feamster, and W. de Donato, "Bismark: A testbed for deploying measurements and applications in broadband access networks," in *Proc. USENIX ATC*, 2014, pp. 383–394.

[22] P. Schmitt, F. Bronzino, R. Teixeira, T. Chattopadhyay, and N. Feamster, "Enhancing transparency: Internet video quality inference from network traffic," in *Proc. Res. Conf. Commun., Inf. Internet Policy*, 2018, pp. 1–12.

[23] D. Y. Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster, "IoT inspector: Crowdsourcing labeled network traffic from smart home devices at scale," in *Proc. ACM IMWUT*, 2020, pp. 1–21.

[24] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the smart home private with smart(er) IoT traffic shaping," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 3, pp. 128–148, Jul. 2019.

[25] (2021). *NAT-PT for IPv6*. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_nat/configuration/15-mt/nat-15-mt-book/ip6-natpt.html

[26] (2021). *RFC 8504: IPv6 Node Requirements*. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8504

[27] Y. Amar, H. Haddadi, R. Mortier, A. Brown, J. Colley, and A. Crabtree, "An analysis of home IoT network traffic and behaviour," 2018, *arXiv:1803.05368*. [Online]. Available: https://arxiv.org/abs/1803.05368

[28] K. Yang, Q. Li, and L. Sun, "Towards automatic fingerprinting of IoT devices in the cyberspace," *Comput. Netw.*, vol. 148, pp. 318–327, Jan. 2019.

[29] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based attacks to discover and control local IoT devices," in *Proc. Workshop IoT Secur. Privacy*, Aug. 2018, pp. 29–35.

[30] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "IoT devices recognition through network traffic analysis," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 5187–5192.

[31] L. Bai, L. Yao, S. S. Kanhere, X. Wang, and Z. Yang, "Automatic device classification from network traffic streams of Internet of Things," in *Proc. IEEE 43rd Conf. Local Comput. Netw. (LCN)*, Oct. 2018, pp. 1–9.

[32] 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 Secur.*, 2006, pp. 16–89.

[33] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "IoTSense: Behavioral fingerprinting of IoT devices," 2018, *arXiv:1804.03852*. [Online]. Available: https://arxiv.org/abs/1804.03852

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

[35] 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 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 2177–2184.

[36] H. Jafari, O. Omotere, D. Adesina, H.-H. Wu, and L. Qian, "IoT devices fingerprinting using deep learning," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 1–9.

[37] S. Aneja, N. Aneja, and M. S. Islam, "IoT device fingerprint using deep learning," in *Proc. IEEE Int. Conf. Internet Things Intell. Syst. (IOTAIS)*, Nov. 2018, pp. 174–179.

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

[39] L. Babun, H. Aksu, L. Ryan, K. Akkaya, E. S. Bentley, and A. S. Uluagac, "Z-IoT: Passive device-class fingerprinting of ZigBee and Z-wave IoT devices," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–7.

[40] A. M. Ali, E. Uzundurukan, and A. Kara, "Assessment of features and classifiers for Bluetooth RF fingerprinting," *IEEE Access*, vol. 7, pp. 50524–50535, 2019.

[41] P. Robyns, E. Marin, W. Lamotte, P. Quax, D. Singelée, and B. Preneel, "Physical-layer fingerprinting of LoRa devices using supervised and zero-shot learning," in *Proc. 10th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Jul. 2017, pp. 58–63.

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

[43] N. J. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic," 2017, *arXiv:1705.06805*. [Online]. Available: https://arxiv.org/abs/1705.06805
[44] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Ping-Pong: Packet-level signatures for smart home device events," 2019, *arXiv:1907.11797*. [Online]. Available: https://arxiv.org/abs/1907.11797
[45] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "HomeSnitch: Behavior transparency and control for smart home IoT devices," in *Proc. 12th Conf. Secur. Privacy Wireless Mobile Netw.*, May 2019, pp. 128–138.
[46] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy, "DEFT: A distributed IoT fingerprinting technique," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 940–952, Feb. 2019.

Xiaobo Ma (Member, IEEE) received the Ph.D. degree in control science and engineering from Xi'an Jiaotong University, Xi'an, China, in 2014. He was a Post-Doctoral Research Fellow with The Hong Kong Polytechnic University. He is currently an Associate Professor with the MOE Key Laboratory for Intelligent Networks and Network Security, Faculty of Electronic and Information Engineering, Xi'an Jiaotong University. He is an XJTU Tang Scholar. His research interests include internet measurement and cyber security.

Jian Qu received the bachelor's degree in computer science and technology in 2019. He is currently pursuing the Ph.D. degree with the MOE Key Laboratory for Intelligent Networks and Network Security, Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China. He was in the Special Class for the Gifted Young at Xi'an Jiaotong University. His current research interests include internet traffic analysis and cyber security.

Jianfeng Li received the Ph.D. degree in control science and engineering from Xi'an Jiaotong University, Xi'an, China, in 2018. He was with the MOE Key Laboratory for Intelligent Networks and Network Security, Faculty of Electronic and Information Engineering, Xi'an Jiaotong University. He is currently a Post-Doctoral Research Fellow with The Hong Kong Polytechnic University. His research interests include network security and privacy.

John C. S. Lui (Fellow, IEEE) received the Ph.D. degree in computer science from UCLA. He is currently Choh-Ming Li Chair Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He is also leading a group of students and post-doctoral students at the Advanced Networking and System Research Laboratory (ANSRLab). His current research interests are in online learning algorithms and applications, machine learning on network sciences and networking systems, large scale data analytics, network/system security, network economics, large scale storage systems, and performance evaluation theory. He is a Fellow of ACM and a Croucher Senior Research Fellow.

Zhenhua Li received the B.S. and M.S. degrees from Nanjing University in 2005 and 2008, respectively, and the Ph.D. degree from Peking University in 2013, all in computer science and technology. He is currently an Associate Professor with the School of Software, Tsinghua University. His research areas cover mobile networking/emulation and cloud computing/storage. He is a Professional Member of IEEE and a Senior Member of ACM.

Wenmao Liu received the Ph.D. degree in information security from Harbin Institute of Technology in 2013. He is currently the Director of the Innovation Center and also a Leader of XingYun Lab, NSFOCUS Inc., and the Co-Chair of Cloud Security Service WG, CSA. During the first two years at NSFOCUS, he was worked as a Post-Doctoral Researcher at Tsinghua University. His research interests are focused on cloud security, the IoT security, data-driven analytics, and other new research areas of network security.

Xiaohong Guan (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the University of Connecticut, Storrs, in 1993. Since 1995, he has been with the Department of Automation, Tsinghua National Laboratory for Information Science and Technology, and the Center for Intelligent and Networked Systems, Tsinghua University. He is currently with the MOE Key Laboratory for Intelligent Networks and Network Security, Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China, where he is also the Dean of the Faculty of Electronic and Information Engineering. He is an Academician of Chinese Academy of Sciences.