Detecting IoT Devices in the Internet

Hang Guo John Heidemann
USC/Computer Science Dept and Information Sciences Institute {hangguo,johnh}@isi.edu

***Abstract*—Distributed Denial-of-Service (DDoS) attacks launched from compromised Internet-of-Things (IoT) devices have shown how vulnerable the Internet is to large-scale DDoS attacks. To understand the risks of these attacks requires learning about these IoT devices: where are they? how many are there? how are they changing? This paper describes three new methods to find IoT devices on the Internet: server IP addresses in traffic, server names in DNS queries, and manufacturer information in TLS certificates. Our primary methods (IP addresses and DNS names) use knowledge of servers run by the manufacturers of these devices. Our third method uses TLS certificates obtained by active scanning. We have applied our algorithms to a number of observations. With our IP-based algorithm, we report detections from a university campus over 4 months and from traffic transiting an IXP over 10 days. We apply our DNS-based algorithm to traffic from 8 root DNS servers from 2013 to 2018 to study AS-level IoT deployment. We find substantial growth (about $3.5\times$) in AS penetration for 23 types of IoT devices and modest increase in device type density for ASes detected with these device types (at most 2 device types in 80% of these ASes in 2018). DNS also shows substantial growth in IoT deployment in residential households from 2013 to 2017. Our certificate-based algorithm finds 254k IP cameras and network video recorders from 199 countries around the world.**

I. Introduction

There is huge growth in sales and the installed base of Internet-of-Things (IoT) devices like Internet-connected cameras, light-bulbs, and TVs. Gartner forecasts the global IoT installed base will grow from 3.81 billion in 2014 to 20.41 billion in 2020 [12].

This large and growing number of devices, coupled with multiple security vulnerabilities, brings an increasing concern about the security threats they raise for the Internet ecosystem. A significant risk is that compromised IoT devices can be used to mount large-scale Distributed Denial-of-Service (DDoS) attacks. In 2016, the Mirai botnet, with over 100k compromised IoT devices, launched a series of DDoS attacks that set records in attack bit-rates. Estimated attack sizes include a 620 Gb/s attack against cybersecurity blog KrebsOnSecurity.com (2016-09-20) [21], and a 1 Tb/s attack against French cloud provider OVH (2016-09-23) [33] and DNS provider Dyn (2016-10-21) [10]. The size of the Mirai botnet used in these attacks has been estimated at 145k [33] and 100k [10]. Source code to the botnet was released [25], showing it targeted IoT devices with multiple vulnerabilities.

If we are to defend against IoT security threats, we must understand how many and what kinds of IoT devices are deployed. Our paper proposes three algorithms to discover the location, distribution and growth of IoT devices. We believe our algorithms and results could help guide the design and deployment of future IoT security solutions by revealing the scale of IoT security problem (how wide-spread are certain IoT devices in the whole or certain part of Internet?), the problem's growth (how quickly do new IoT devices spread over the Internet?) and the distribution of the problem (which countries or autonomous systems have certain IoT devices?). Our goal here is to assess the scope of the IoT problem; improving defenses is complementary future work.

Our IoT detection algorithms can also help network researchers study the distribution and growth of target IoT devices and help IT administrators discover and monitor IoT devices in their network. As more every-day objects get connected into the Internet, our algorithms may even help understand the physical world by, for example, detecting and tracking network-enabled vehicles for crime investigation.

Our first contribution is to propose three IoT detection methods. Our two main methods detect IoT devices from observations of network traffic: IPs in Internet flows (§II-A2) and stub-to-recursive DNS queries (§II-A3). They both use knowledge of servers run by manufacturers of these devices (called *device servers*). Our third method detects IoT devices supporting HTTPS remote access (called *HTTPS-Accessible IoT devices*) from the TLS (Transport Layer Security [8]) certificates they use (§II-B). (We reported an early version of IP-based detection method [18]; here we add additional methods and better evaluate our prior method in §III-A2.)

Our second contribution is to apply our three detection methods to multiple real-world network measurements (Table II). We apply our IP-based method to flow-level traffic from a college campus over 4 months (§III-A2) and a regional IXP (Internet Exchange Point [6]) over 10 days (§III-A3). We apply our DNS-based method to DNS traffic at 8 root name servers from 2013 to 2018 (§III-B1) to study IoT deployment by Autonomous Systems (ASes [22]). We find about $3.5\times$ growth in AS penetration for 23 types of IoT devices and modest increase in device type density for ASes detected with these device types (we find at most 2 known device types in 80% of these ASes in 2018). We confirm substantial deployment growth at household-level by applying DNS-based method to DNS traffic from a residential neighborhood from 2013 to 2017 (§III-B2). We apply our certificate-based method to a public TLS certificate dataset (§III-C) and find 254K IP cameras and network video recorders (NVR) from 199 countries.

This paper builds on prior work in the area. We draw on data from University of New South Wales (UNSW) [38]. Others are currently studying the privacy and vulnerabilities of individual devices (for example [1]); we focus on detection. Prior work has studied detection [37], [38], [36], [9], [3], [5], [28], but we use different detection signals to observe devices behind NATs (Network Address Translations devices [11]) as well as those on public IP addresses (detailed comparisons in §V). We published an early version of IP-based detection in a workshop [18]. This paper adds two new detection methods: DNS-based detection (§II-A3) and certificate-based detection (§II-B) and adds a new 4-month study of IoT devices on college campus for IP-based detection (§III-A2).

Our studies of IP-based and DNS-based detections are approved by USC IRB as non-human subject research (IRB IIR00002433 on 2018-03-27 and IRB IIR00002456 on 2018-04-19). We make data captured from our 10 IoT devices (Table I) public at [16].

II. Methodology

We next describe our three methods to find IoT devices. Our three detection methods use different types of network measurements (IPs in Internet flows §II-A2, stub-to-recursive DNS queries §II-A3 and TLS certificates §II-B) to achieve different coverage of IoT devices. Combining our three methods reveals a more complete picture of IoT deployment in the Internet. (However, even with all three methods, we do not claim complete coverage of global IoT deployment.)

A. IP and DNS-Based Detection Methods

Our two main methods detect general IoT devices from two types of passive measurements: Internet flows, measured from any vantage point in the Internet (IP-based method); and DNS queries, measured between stub and recursive servers (DNS-based method). These two methods cover IoT devices that are visible to these two data sources, including those that use public IP addresses or are behind NAT devices.

Our methods exploits the observation that most IoT devices exchange traffic regularly with device-specific servers. (For example, IoT inspector project observes 44,956 IoT devices from 53 manufactures talking to cloud servers during normal operation [20].) If we know these servers, we can identify IoT devices by watching traffic for these packet exchanges. Since servers are usually unique for each class of IoT device, we can also identify the types of devices. Our approaches consider only with whom IoT devices exchange traffic, not patterns like

Manufacturer	**Model**	**Alias**
Amazon	Dash Button	Amazon_Button
Amazon	Echo Dot	Amazon_Echo
Amazon	Fire TV Stick	Amazon_FireTV
Amcrest	IP2M-841 IP Cam	Amcrest_IPCam
D-Link	DCS-934L IP Cam	D-Link_IPCam
Foscam	FI8910W IP Cam	Foscam_IPCam
Belkin (Wemo)	Mini Smart Plug	Belkin_Plug
TP-Link	HS100 Smart Plug	TPLink_Plug
Philips (Hue)	A19 Starter Kit	Philips_LightBulb
TP-Link	LB110 Light Bulb	TPLink_LightBulb

TABLE I: The 10 IoT Devices that We Purchased

timing or rates, because patterns are often obscured when traffic mixes (such as with multiple devices behind a NAT).

Our two methods depend on identifying servers that devices talk to (§II-A1), and looking for these servers by IP address (§II-A2) and DNS name (§II-A3).

Although our method is general, it requires knowledge of what servers devices talk to, and therefore it requires device-specific data obtained by us or others. We still detect devices that change the servers with which they interact provided they continue to talk to most of their old servers. For IoT devices behind NAT, our methods only identify the existence of each type of IoT devices but can not know the exact number of devices for each type because we cannot count NATted devices outside the NAT.

1) Identifying Device Server Names: Our approach depends on knowing what servers devices talk to. Our goal is to find domain names for all servers that IoT devices regularly and uniquely talk to. However, we need to remove server names that are often shared across multiple types of devices, since they would otherwise produce false detections.

Note that even with our filtering of common shared server names, we sometimes find servers that are shared across multiple types of devices. We handle this ambiguity from shared servers by not trying to distinguish these devices types in detection, as we explain later in this section.

Identifying Candidate Server Names: We bootstrap our list of candidate server names by purchasing samples of IoT devices and recording who they talk to. We describe the list of devices we purchased in Table I and provide the information we learned as a public dataset [16].

For each IoT device we purchase, we boot it and record the traffic it sends. We extract the domain name of server candidates from type A DNS requests made by target IoT device in operation. We capture DNS queries at the ingress side of recursive DNS resolver to mitigate effects of DNS caching.

Filtering Candidate Server Names: We exclude domain names for two kinds of servers that would otherwise cause false positives in detection. One is *third-party servers*: servers not run by IoT manufacturers that are often shared across many device types. The other is *human-facing servers*: servers that also serve human.

Third-party servers usually offer public services like time, news and music streaming and video streaming. If we include them, they would cause false positives because they interact many different clients.

We consider server name S as a third-party server for some IoT product P if neither P's manufacturer nor the sub-brand P belongs to (if any) is a substring of S's domain (regardless of case). We define domain of a URL as the immediate left neighbor of the URL's public suffix. (We identify public suffix based on public suffix list from Mozilla Foundation [30]). We use Python library tldextract to identify TLD suffixes [23].

Human-facing servers serve both human and device (note that all server candidates serve device because they are DNS queried by IoT devices in the first place). They may cause mis-classifying a laptop or cellphone (operated by human) as IoT devices.

We identify human-facing servers by if they respond to web requests (HTTP or HTTPS GET) with human-focused content. We define *respond* as returning an HTML page with status code 200. We define *human-focused content* as the existence of any web content instead of place-holder content. Typically place-holder content is quite short. (For example, http://appboot.netflix.com shows place holder "Netflix appboot" and is just 487 bytes.) So we treat HTML text longer than 630 bytes as human-focused content. We determined this threshold empirically from HTTP and HTTPS content at 158 server domain names queried by our 10 devices (Table I).

We call the remaining server names *device-facing manufacturer server*, or just *device servers*, because they are run by IoT manufacturers and serve devices only. We use device servers for detection.

Handling Shared Server Names: Some device server names are shared among multiple types of IoT devices from the same manufacturer and can cause ambiguity in detection.

If different device types share the exact set of server names, then we cannot distinguish them and simply treat them as the same type—a *device merge*.

If different device types have partially overlapping sets of device server names, we can not guarantee they are distinguishable. If we treat them as separate types, we risk false positives and confusing the two types. We avoid this problem with *detection merge*: when we detect device types sharing common server names, we conservatively report we detect at least one of these device types. (Potentially we could look for unique device servers in each type; we do not currently do that.)

Handling Future Server Name Change: The server names that our devices (Table I) use are quite stable over 1 to 1.5 years (as shown in §IV-B). However, both our IP-based and DNS-based detection risks missing devices that get software updates that cause them to talking to new server names. We mitigate these potential missed detections by reporting that a device exists when we see a majority of server names for that device (both IP-based method §II-A2 and DNS-based method §II-A3). For DNS-based method, we also propose a technique to discover new device server names during detection (§II-A3).

2) IP-Based IoT Detection Method: Our first method detects IoT devices by identifying packet exchanges between IoT devices and device servers. For each device type, we track *device-type-to-server-name mapping*: a list of device server names that type of devices talks to. We then define a threshold number of server names; we interpret the presence of traffic to that number of server names (identified by server IP) from a given IP address as indicating the presence of that type of IoT device.

Tracking Server IP Changes: We search for device servers by IP addresses in traffic, but we discover device servers by domain names in sample devices. We therefore need to track when DNS resolution for server name changes.

We assume server names are long-lived, but the IP addresses they use sometimes change. We also assume server-name-to-IP mappings could be location-dependent.

We track changes of server-name-to-IP mapping by resolving server names to IP addresses every hour (frequent enough to detect possible DNS-based load balancing). To make sure IPs for detection are correct, we track server IPs across the same time period and at roughly the same geo-location as the measurement of network traffic under detection.

Completeness Threshold Selection: Since some device servers may serve both devices and individuals (due to we use necessary condition to determine device-facing server in §II-A1 and risk mis-classifying human-facing manufacturer server as device server) and sometimes we might miss traffic to a server name due to observation duration or lost captures, we set a threshold of server names required to indicate the presence of each IoT device type. This threshold is typically a majority, but not all, of the server names we observe a representative device talk to in the lab. (This majority-but-not-all threshold also mitigates potential detection misses caused by devices that start talking to new servers.)

Most devices talk to a handful of device server names (up to 20, from our laboratory measurements §III-A1). For these types of devices, we require seeing at least $2/3$ device server names to believe a type of IoT device exists at a given source IP address. Threshold $2/3$ is chosen because for devices with 3 or more server names, requiring seeing anything more than $2/3$ server names will be equivalent to requiring seeing all server names for some devices. For example, requiring at least $4/5$ server names is equivalent to requiring all server names for devices with 3 to 4 device server names.

For devices that talk to many device server names (more than 20), we lower our threshold to $1/2$. Typically these are devices with many functions and the manufacturer uses a large pool of server names. (For example, our Amazon_FireTV, as in Table I, has 41 device server names.) Individual devices will most likely talk to only a subset of the pool, at least over short observations.

Limitation: Although effective, IP-based detection faces two limitations. First, it cannot detect IoT devices in previously stored traces, since we usually do not know device server IPs in the past, and coverage of commercial historical DNS datasets can be limited ([18]). Second, we assume we can learn the set of servers the IoT devices talk to. If we do not learn all servers during bootstrapping (§II-A1), or if device behavior changes (perhaps due to a firmware update), we need to learn new servers. However we cannot learn new device servers during IP-based detection because we find it hard to judge if an unknown IP is a device server, even with help of reverse DNS and TLS certificates from that IP. These limitations motivate our next detection method.

3) DNS-Based IoT Detection Method: Our second method detects IoT devices by identifying the DNS queries prior to actual packet exchanges between IoT devices and device servers.

Strengths: This method addresses the two limitations for IP-based detection (§II-A2). First, we can directly apply DNS-based detection to old network traces because server names are stable while server IP can change. Second, we can learn new device server names during DNS-based detection by examining unknown server names DNS queried by detected IoT devices and learning those look like device servers (using rules in §II-A1).

Limitations: This method requires observation of DNS queries between end-user machines and recursive DNS servers, limiting its use to locations that can see "under" recursive DNS revolvers. This method also works with recursive-to-authority DNS queries (see §III-B) when observations last longer than DNS caching, since then we see users-driven queries for server names even above the recursive. Detection with recursive-to-authority DNS queries reveals presence of IoT devices at the AS-level, since recursives are usually run by ISPs (Internet service providers [39]) for their users.

Method Description: Our DNS-based method has three components: *detection*, *server learning* and *device splitting*. Figure 1 illustrates this method's overall workflow: it repeatedly conducts detections with the latest knowledge of IoT device server names, learns new device server names after each detection, and terminates when no new server names are learned (see the loop of "Detection" and "Server Learning" in Figure 1). This method also revises newly learned server names by device splitting if it suspects they are incorrect, as signaled by decreased detection after new server names are added (see "Device Splitting" in Figure 1).

Detection: Similar to §II-A2, for each type of IoT devices, we track a list of device server names that type of device talks to. We interpret presence of DNS queries for above a threshold (same as §II-A2) amount of device server names from a give IP address as presence of that IoT device type. (We call this IP *IoT user IP*.)

To cover possible variants of known device servers, in detection, we treat digits in server name's sub-domain as matching any digit. We define sub-domain of a URL as everything on the left of the URL's domain (URL's domain as defined in §II-A1).

Server Learning: After each detection, we learn new device server names and use them in subsequent detections. Specifically, we examined unknown server names DNS queried by IoT user IPs and if we find any unknown server names resemble device servers for certain IoT device detected at certain IoT user IP (judged by rules in §II-A1), we extend this IoT device' server name list with these unknown server names.

Device Splitting: We may incorrectly merge two types of devices that talk to different set of servers if we only know their shared server names prior to detection.

Incorrect device merges can reduce detection rates. When we falsely merge different device types $P1$ and $P2$ as P, we risk learning new server names for the merged type P that $P1$ and $P2$ devices do not both talk to and causing reduced detections of P in subsequent iterations because we miss some $P1$ (or $P2$) devices by searching for the newly-acquired server names that $P1$ (or $P2$) do not talk to.

Device splitting addresses this problem by reverting incorrect merge. If we detect fewer device types P at certain IP after learning new server names, we know P is an incorrect merge of two different device types, $P1$ and $P2$, and that the new server names learned for P do not apply for both $P1$ and $P2$. We thus split P into $P1$ and $P2$, with $P1$ talking to P's server names before last server learning (without newly-learned server names) and $P2$ talk to P's latest server names (with the new server names). We show an example of how device splitting reverts an incorrect device merge later in controlled experiment (§IV-B).

B. Certificate-Based IoT Detection Method

Our third method detects IoT devices using HTTPS by active scanning for TLS certificates and identifying target IoT devices' TLS certificates. This method thus covers HTTPS-Accessible IoT devices either with public IPs or behind NATs but forwarded to a public port. However, certificate scanning will miss devices behind NATs that lack public-facing IP addresses and IoT devices that do not use TLS

Note that prior work has mapped TLS certificate to IoT devices, both by matching text (like "IP camera") with certificates [36], and by using community-maintained annotations [9]. In comparison, our method uses multiple techniques to improve the accuracy of certificate match-

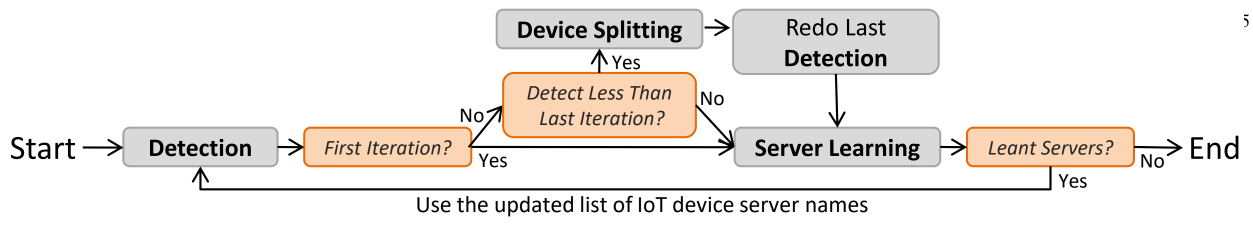

Fig. 1: Workflow for DNS-Based IoT Detection with Server Learning

ing, and also confirms that matched certificates come from HTTPS servers running in IoT devices.

We use existing public crawls of IPv4 TLS certificates. We first identify *candidate certificates*: the TLS certificates that contain target devices' manufacturer names and (optionally) product information. Candidate certificates most likely come from HTTPS servers related to target devices such as HTTPS servers ran by their manufacturers and HTTPS servers ran directly in them. We then identify *IoT certificates*: the candidate certificates that come from HTTPS servers running directly in target devices. Each IoT certificate represents a HTTPS-Accessible IoT device.

1) Identify Candidate Certificates: We identify candidate certificates for every target device by testing each TLS certificate against a set of text strings we associate with each device (called *matching keys*). (We describe where our list of target devices is found in §III-C.)

Matching Keys: We build a set of matching keys for each target device with the goal to suppress false positives in finding candidate certificates. If a target device's manufacturer does not produce any other type of Internet-enabled products (per product information on manufacturer websites), its matching key is simply the name of its manufacturer (called *manufacturer key*). Otherwise, its matching keys will be manufacturer key plus its product type (like "IP Camera"). We also include IoT-specific sub-brands (if any). For example, "American Dynamics" is the sub-brand associated the IP cameras manufactured by Tyco International.

We do two kinds of matching between a matching key K and a field S in TLS Certificate: *Match* means K is a substring of S (ignore case); *Good-Match* means K is a Match of S and the character(s) adjacent to K's match in S are neither alphabetical nor numbers. For example, "GE" is a Match but not a Good-Match of "Privilege" because the adjacent characters of "GE" in "Privilege" is "e" (an alphabet). (We do not simply look for identical K and S because often S uses a prefix or suffix. For example, a certificate's subject-organization field "Amcrest Technologies LLC" will be a Good-Match with manufacturer key "Amcrest", but is not identical due to the suffix "Technologies LLC".)

Requiring Good-Match for manufacturer keys reduces false positives caused by IoT manufacturer names being substrings of other companies. For example, name of IP camera manufacturer "Axis_Communications" is a substring of Telecom company "Maxis_Communications" but they are not a Good-Match.

We use the Match (not Good-Match) rule for other keys (product types and sub-brand) because they require greater flexibility. For example, product type "NVR" can be matched to text string like "myNVR".

Key Matching Algorithm: We test each TLS certificate (input) with matching keys from each target device. Specifically, we examine four subject fields in a TLS certificate C (organization C_O, organization units C_{OU}, common name C_{CN} and SubjectAltNames C_{DN}, if present) and consider C a candidate certificate for device P if P's manufacturer key (K_m^P) Good-Matches C_O and any non-manufacturer keys for P Match any of these four subject fields in C.

We handle two edge cases when testing if K_m^P Good-Matches C_O. If C_O is empty, or an default ("SomeOrganization" or "company"), we instead test if K_m^P Good-Matches any of the other three fields we examine (C_{OU}, C_{CN} and C_{DN}). If we compare K_m^P to a field that is a URL, we only match K_m^P against the URL's domain part (URL's domain as defined in §II-A1) because domain shows ownership of a server name. (For example, Accedo Broadband owns *.sharp.accedo.tv'', not Sharp.)

2) Identify IoT Certificate: We identify IoT-specific certificates because they are not typically signed by a certificate authority (CA). We identify them because they are self-signed and lack valid domain names.

Self Signing: Many HTTPS servers on IoT devices use self-signed certificates rather than CA-signed certificates to avoid the cost and complexity of CA-signing. We consider a candidate certificate C (for device P) self signed if C's issuer organization C_{iO} is either a copy of any of the 4 subject fields we examined (C_O, C_{OU}, C_{CN} and C_{DN}) or is Good-Matched by P's manufacturer key (K_m^P).

Lacking Valid Domain Names: Often IoT users lack dedicated DNS domain names for their home network. The only exception we found is some devices use "www."+manufacter+".com" as a place holder for C_{CN}. (For example, www.amcrest.com for Amcrest IP Camera.)

We consider a candidate certificate C lacking valid domain names if none of the values in C_{CN} and C_{DN} (if present) is a valid domain name. We ignore Dynamic DNS names (using a public list of dynamic DNS providers [32]) and default names.

C. Adversarial Prevention of Detection

Although our methods generally work well in IoT detection, they are not designed to prevent an adversary

Dataset	Type	Span	IP Assignment	Coverage
USC	IP	4 Months	Dynamic	A College Campus
FRGP	IP	10 Days	Dynamic	An IXP's Clients
DITL	DNS	6 Years	N/A	The Whole Internet
CCZ	DNS	5 Years	Static	A Neighborhood
ZMap	Cert	1 Day	N/A	The Public Internet

TABLE II: Datasets for Real-world IoT Detection

from hiding IoT devices. For example, use of a VPN (Virtual Private Network [13]) that tunnels traffic from the IoT to its servers would evade IP-based detection. IoT devices that access device servers with hard-coded IP addresses rather than DNS names will avoid our DNS-based detection. Although an adversary can hide IoT devices, since they are designed for consumer use and to minimize costs, we do not anticipate widespread intentional concealment of IoT devices. (We did not observe any devices intentionally avoiding detection during our study)

III. Results: IoT devices in the Wild

We next apply our detection methods with real-world network traffic (Table II) to learn about the distribution and growth of IoT devices in the wild.

Although ground truth for the entire Internet is impossible to get, we demonstrate our methods show high detection accuracy in controlled experiments with controlled ground truth in §IV, and we evaluate at our university and with an IXP next.

A. IP-Based IoT Detection Results

To apply our IP-based detection, we first extract device server names from 26 devices by 15 vendors (§III-A1). We then apply detection to Internet flows at a college campus from a 4-month period (§III-A2) and partial traffic from an IXP (§III-A3).

1) Identifying Device Server Names: We use device servers from two sets of IoT devices in detection: 10 IoT devices we purchased (Table I) and 21 IoT devices from data provided by the UNSW (devices as listed in Figure.1b of [38]). (Our 10 devices were chosen for their popularity on amazon.com in 2018.) We extract device server names from both sets of devices with method in §II-A1.

We break-down server names we found. Of the 171 candidate server names from our 10 devices, about half (56%, 96) are third-party servers, providing time, news or music streaming, while the other half (44%, 75) are manufacturer servers. Of these manufacturer servers, only a small portion (7%, 5) are human-facing (like prime.amazon.com). The majority of manufacturer servers (93%, 70) are device-facing and will be used in detection.

We manually examine the 171 candidate server names and confirm the classifications for most of them are correct (for 157 out of 171, ownership of server domain is verified by whois or websites).

We cannot verify ownership of 11 candidate server names. Luckily, our method lists them as third-party servers and they will not be used in detection. We find three candidate server-names (api.xbcs.net, heartbeat.lswf.net, and nat.xbcs.net) are falsely classified as third-party servers. We confirm they are run by IoT manufacturer Belkin based on "whois lswf.net" and prior work [34] and add them back to our list. These three server names fail our test for manufacturer server (§II-A1) because their domains show no information of manufacturer.

Similarly, we extracted 48 device servers from 18 of 21 IoT devices from UNSW (using datasets available on their website https://iotanalytics.unsw.edu.au). The remaining 3 of their devices are not detectable with our method because they only visit third-party and human-facing servers.

Combining server names measured from our 10 devices and the 18 detectable devices from UNSW (merging two duplicate devices, Amazon_Echo and TPLink_Plug) gives us 26 detectable IoT devices; Among these 26 detectable IoT devices, we merge TPLink_IPCam, TPLink_Plug and TPLink_Lightbulb as a meta-device because they talk to the same set of of device servers (a device merge, recall in §II-A1). Similarly, we merge Belkin_Switch and Belkin_MotionSensor. After device merge, we are left with 23 merged devices talking to 23 distinct sets of device server names. (Together they have 99 distinct device server names.)

By detecting with these server names, we are essentially looking for 23 types of IoT devices that talk to these 23 set of server names, including but not limited to the 26 IoT devices owned by us and UNSW.

2) IoT Deployment in a College Campus: We apply our IP-based detection method to partial network traffic from our university campus for a 4-month period in 2018.

Input Datasets: We use passive Internet measurements at the University of Southern California (USC) guest WiFi for 4 different 4-day-long periods from August to November in 2018 (Table II). To protect user privacy, packet payloads are not kept and IPs are anonymized by scrambling the last byte of each IP address in a prefix preserving manner.

Input Server IPs: Since server-name-to-IP bindings could vary over time and physical locations (as discussed in §II-A2), we collect latest IPv4 addresses for our 99 device server name daily at USC, as described in §II-A1. Ideally we would always use the latest server IPs in detection. However, due to outages in our infrastructure, we can ensure the server IPs we use in detections are no more than one-month old.

IoT Detection Results: As shown in Table III, IoT detections increase on campus from August to September (from 13 to 23), but decrease in October and November (to 19 and then 10). In comparison, IoT user IPs on campus remain the same from August to October (6) and drop in November (3). (We discuss reasons behind

Month	IoT Detection	IoT User IP	Est IoT Users (Res : Non-Res)	Est IoT Devices
Aug	13	6	2 (2 : 0)	5 to 7
Sep	23	6	5 (2 : 3)	21 to 28
Oct	19	6	4 (3 : 1)	11 to 15
Nov	10	3	2 (2 : 0)	8 to 12

TABLE III: 4-Month IoT Detection Results on USC Campus and Our Estimations of IoT Users and Devices

IP-A & IP-H	IP-B	IP-C & IP-F	IP-D
LiFX_LightBulb	Withings_*	HP_Printer	LiFX_LightBulb
	Amazon_*	Withings_*	Withings_*
		Amazon_*	Amazon_*

TABLE IV: August IoT Detection Results on USC Campus (Merging IPs with Identical Detections)

these variations in campus IoT deployment later in this section.)

We show our August detection results in Table IV. (Detections in other months are similar.) Note that "Amazon_*" in Table IV stands for at least one of Amazon_FireTV and Amazon_Echo. Similarly "Withings_*" stands for at least one of Withings_Scale and Withings_SleepSensor (recall detection merge in §II-A1). We find that IoT user IPs are often detected with multiple device types, suggesting the use of network-address translation (NAT) devices. We also find two sets of IoT user IPs (A and H; C and F) , each sharing the exact set of IoT device types. A likely explanation is these two sets of IPs belong to two IoT users using dynamically assigned IP addresses, and these addresses change one time during our 4-day observation. (More discussions of IoT users on campus later.)

Since USC guest WiFi dynamically assigns IPs, our counts of IoT detections and IoT user IPs risk over-estimating actual IoT deployments on campus. When one user gets multiple IPs, our IoT user IP count over-estimates IoT user count. When one user's devices show up in multiple IPs, our IoT detection count gets inflated. (We validate our claim that dynamic IPs inflate detection in §IV-A.)

Estimating Numbers of IoT Users and Devices: To get a better knowledge of actual IoT deployments on campus, we estimate the number of IoT users on campus based on the insight that although one user could get assigned different IPs, he may still be identified by the combination of IoT device types he owns. We then infer the number of IoT devices we see on campus given this many users.

We infer the existence of IoT users by clustering IoT user IPs from the same month or adjacent months that have similar detections. We consider detections at two IPs (represented by two sets of detected IoT device types $d1$ and $d2$, without detection merge) to be similar if they satisfy the following heuristic: $size(intersect(d1, d2))/size(union(d1, d2)) \geq 0.8$.

While our technique risks under-estimating the number of IoT users by combining different users who happen to own same set of device types into one user, we argue this error is unlikely because most IP addresses that have IoT devices (16 out of 21, 76%) show multiple device types (at least 4, without detection merge), and the chance that two different users have identical sets of device types seems low.

We find three clusters of IPs: with one each spanning 4, 3 and 2 months. These three clusters of IPs likely belong to three *campus residents* who could install their IoT devices relatively permanently on campus, such as students living on campus and faculty (or staff) who have office on campus.

We find four IPs that do not belong to any clusters. These four IPs likely belong to four *campus non-residents* who only brought their devices to campus briefly, such as students living off-campus and other campus visitors.

We then estimate number of IoT devices on campus in each month by adding up devices owned by estimated IoT users in each month. We estimate devices owned by a user in a given month by taking the union of device types detected from this user's IPs in this month and assuming this user owns exactly one device from each detected type. (Recall from §II-A that for NATted IoT devices, our method only identifies the existence of device types but cannot know the device count for each type.)

We summarize our estimated numbers of IoT users and devices in Table III. (Our estimated IoT device counts are ranges of numbers because we do not always know the exact number of detected device types due to detection merge). Our first observations is campus residents are mostly stable except an existing resident disappear in November (likely due to he stops using his only detected device type: LiFX_LightBulb) and a new resident show up in October (likely due to a faculty or staff installing new IoT devices in their office).

Our second observation is number of campus non-residents differs a lot by month. While we find 3 non-residents in September and 1 non-resident in October, we find none in August and November. One explanation for this trend is there are more campus events in the middle of the semester (September and October) which attracts more campus visitors (potentially bringing IoT devices).

We argue that the small number of IoT users and devices we detect is an under-estimation of the actual campus IoT deployment since our measurements only cover campus guest WiFi and we expect IoT devices to be deployed on wired networks and secure WiFi that we do not cover.

3) IoT Devices at an IXP: We also apply IP-based detection to partial traffic from an IXP, using FRGP-ContinuousFlowData (FRGP) dataset [41] collected by Colorado State University from 2015-05-10 to 2015-05-19 (10 days), as in Table II. We find 122 triggered detections of 10 to 11 device types (we do not know exact number of types due to detection merge §II-A1) from 111 IPs. (Similar to §III-A2, since clients of FRGPs may use dynamically assigned IPs, our detection counts and IoT user IPs counts risk being inflated.) Please see our tech report for details [17].

B. DNS-Based IoT Detection Results

We next apply our DNS-based detections to two real-world DNS datasets.

1) Global AS-Level IoT Deployments: We apply detection to Day-in-the-Life of the Internet (DITL) datasets from 2013 to 2018 to explore growth of AS-level deployments for our 23 device types.

Input Datasets: our detection uses DITL datasets from 8 out of 13 root DNS servers (each a *root letter*) between 2013 and 2018 (excluding G, D, E and L roots for not participating in all these DITL data and I root for using anonymized IPs) to show growth in AS-level IoT deployment in this period, as summarized in Table II. Each DITL dataset contains DNS queries received by a root letter in a 2-day window.

Since root DNS servers see requests from recursive DNS resolvers (usually run by ISPs for their users), our results detect devices at the AS-level, not for households. Our results thus show existence of device types in ASes. (They do not show exact device counts, as described before §II-A.) To find out the ASes where detected devices come from, we map recursive DNS resolvers' IPs to AS numbers (ASN) with CAIDA's Prefix to AS mappings dataset [4].

Since the data represents ASes and instead of households, we do detection only (§II-A3) and omit the server-learning portion of our algorithm. With many households mixed together, AS-size aggregation risk learning wrong servers. To count per-device-type detections, we do not use detection merge (§II-A1).

With more than half of all 13 root letters (62%, 8 out of 13), we expect to observe queries from the majority of recursives in the Internet because prior work has showed that under 2-day observation, most (at least 80%) recursives query multiple root letters (with 60% recursives query at least 6 root letters) [31]. However, even with visibility to the majority of recursives, our detection still risks under-estimating AS-level IoT deployment because the 2-day DITL measurement is too short to observe DNS queries from all known IoT device types behind these visible recursives. (Under short observation, IoT DNS queries could be hidden from root letters by both DNS caching and *non-IoT overshadowing*: if a non-IoT device queries a TLD before an IoT device behind the same recursive does, the IoT DNS query, instead of being sent to a root letter, will be answered by the DNS caches created or renewed by the non-IoT DNS query.) Consequently, we mainly focus on the trend shown in our detection results instead of the exact number of detections.

Growth in AS Penetrations: We first study the "breadth" of AS-level IoT deployment by examining the number of ASes that our 23 IoT device types have penetrated into.

We show overall AS penetration for our 23 IoT device types (number of ASes where we find at least of one of our 23 IoT device types) in Figure 2 as the blue crosses. We find the overall AS penetration for our device types increases significantly from 2013 to 2017 (from 244 to 846 ASes, about 3.5 times) but plateau from 2017 to 2018 (from 846 to 856 ASes).

We believe the reason that overall AS penetration for our 23 IoT device types plateau between 2017 and 2018 is the sales and deployment decline as these models replaced by newer releases. To support this hypothesis, we estimate release dates for our device types and compare these estimated release dates with per-device-type AS penetration (number of ASes where each of our 23 device types is found) from 2013 to 2018 (Figure 5).

We estimate release dates for 22 of our 23 device types based on estimated release dates for our 26 detectable IoT devices (recall §II-A1). (We exclude device type HP_Printer here because there are many HP wireless printers released from a wide range of years and it would be inaccurate to estimate release date of this whole device type based on any HP_Printer devices.) If a device type includes more than one of our 26 detectable IoT devices (due to device merge), we estimate release dates for all these devices and use the earliest date for this device type. We estimate release date for a given IoT device from one of three sources (ordered by priority high to low): release date found online, device's first appearance date and device's first customer comment date on Amazon.com. We confirm all the 22 device types are released at least two years before 2017 (2 in 2011, 7 in 2012, 3 in 2013, 5 in 2014 and 5 in 2015), consistent with our claim that their sales are declining in 2017.

We compare estimated release dates with per-device-type AS penetration results (Figure 5) and find that detections of device types tend to plateau after release, consistent with product cycles and a decrease in sales and use of these devices. For example, Withings_SmartScale and Netatmo_WeatherStation, which are released in 2012, stop growing roughly after 2016-10-04 and 2017-04-11, suggesting a product cycle of about 4 and 5 years. In comparison, TPLink-IPCam/Plug/LightBulb is the only device type released around 2016 (TPLink_IPCam on 2015-12-15, TPLink_Plug on 2016-01-01 and TPLink_Lightbulb on 2016-08-09) and their AS penetration continue to rise even on 2018-04-10, despite AS penetration of other device types (released between 2011 and 2015) roughly stop increasing by 2017.

Note the fact that the AS penetrations of our 23 device types plateau does not contradict with the constant growth of overall IoT deployment because new IoT devices are constantly appearing.

Growth in Device Type Density: Having showed that our 23 IoT device types penetrate into about 3.5 times

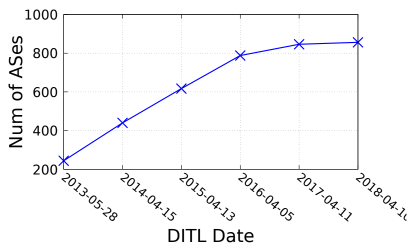

Fig. 2: Overall AS Penetration for Our 23 Device Types from 2013 to 2018

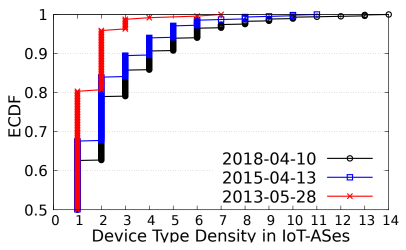

Fig. 3: ECDF for Device Type Density in IoT-ASes from 2013 to 2018

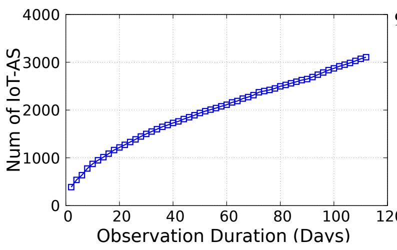

Fig. 4: Detected IoT-ASes under Extended Observation at B Root

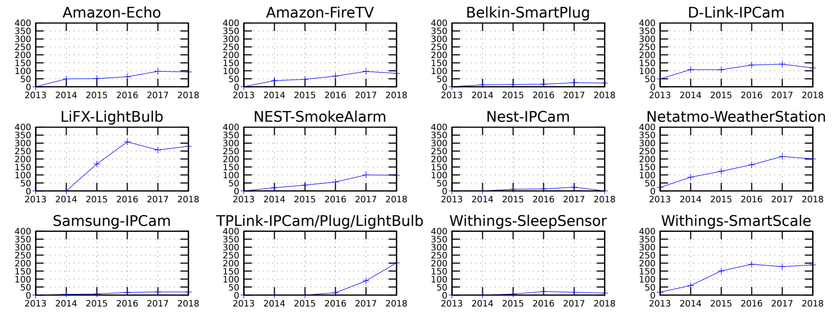

Fig. 5: Per-Device Type AS Penetrations (Omitting 7 Device Types Appearing in Less Than10 ASes)

more ASes from 2013 to 2018, we next study how many IoT device types are found in these ASes—their *device type density*. We use device type density to show the "depth" of AS-Level IoT Deployment.

For every AS detected with at least one of our 23 IoT device types (referred to as *IoT-AS* for simplicity) from 2013 to 2018, we compute its device type density. We present the empirical cumulative distribution (ECDF) for device type densities of IoT-ASes from 2013 to 2018 in Figure 3.

Our first observation from Figure 3 is from 2013 to 2018, not only are there 3.5 times more IoT-ASes (as shown by AS penetration), the device type density in these IoT-ASes are also constantly growing.

Our second observation is despite the constant growth, device type density in IoT-ASes are still very low as of 2018. In 2018, most (79%) of the IoT-ASes have at most 2 of our 23 device types, which is a modest increase comparing to 2013 where the similar percentage (80%) of IoT-ASes have at most 1 of our 23 device types.

Our results suggest that for IoT devices, besides potential to further grow in AS penetration (which would lead to growth in household penetration), there exists even larger potential to grow in device type density (which would lead to growth in device density). This unique potential of two-dimensional growth (penetration and density) sets IoT devices apart from other fast-growing electronic products in recent history such as cell-phone and personal computer (PC) which mostly grow in penetration (considering that while a person may only own 1 to 2 cell-phones and PCs, he could own many more IoT devices).

We rule out the possibility that the increasing AS penetration and device type density we observe is an artifact of device servers we used in detection (measured around 2017) do not apply to IoT devices in the past by showing IoT device-type-to-server-name mappings are stable over time in §IV-B.

ASes with Highest Device Type Density in 2018: We examined the top 10 ASes with highest device type density in 2018 (detected with 8 to 14 of our 23 device types). Our first observation is that they are pre-dominantly from the U.S. (4 ASes) and Europe (3 ASes). There are also 2 ASes from Eastern Asia (Korea and China) and 1 from Haiti. This distribution also consistently show up in top 20 ASes with 10 ASes from the U.S. and 5 ASes from Europe. Our second observation is that these top 10 ASes are mostly major consumer ISPs in their operating regions such as Comcast, Charter, AT&T and Verizon from the U.S., Korea Telecom from South Korea and Deutsche Telekom for Germany.

Estimating Actual Overall AS Penetration in 2018: Recall that the overall AS penetrations for our 23 device types reported in Figure 2 are under-estimations of the ground truth, because our DITL data is not complete (8 of 13 root letters provide visibility to most but not all global recursives), and because two-day data will miss many queries due to DNS caching and non-IoT overshadowing.

We estimate actual overall AS penetration in 2018 by applying detection to extended measurement at B root. With this extended measurement, we expect to observe queries from most global recursives at B root because most global recursives rotate among root letters (at least 80% [31]). We also hope to observe IoT DNS queries that would otherwise get hidden by DNS caching and non-IoT overshadowing in short observation. (Ideally, when adding more observations leads to no new detections, we know we have detected all IoT-ASes that could be visible to B root.)

To evaluate how many IoT-ASes we could see, we extend 2-day 2018 DITL observation at B root to 112 days. As shown in Figure 4, we see a constant increase in detection of IoT-ASes over longer observation. With 112-day observation, we detect 3106 IoT-ASes, 8× more than what we see in 2 days of B root only (388 IoT ASes), and 3.6× more than 2 days with 8 roots (856 IoT ASes, as in Figure 2). In 112 days, we see about 5% of all unique ASes in the routing system in early 2018 (about 60,000, reported by CIDR-report.org [42]) However, we do not see the detection curve in Figure 4 flattening even after 112 days.

We model IoT query rates from an IoT-AS as seen by a single root letter. Simple models (a root letter receives 1/13th of the traffic) show a curve flattening after at least 300 days, consistent with what we see in Figure 4. However, a detailed model requires understanding the IoT query rates and the aggregate (IoT and non-IoT) query rates, more information than we have. We conclude that the real numbers of IoT-ASes are much higher than our detections with DITL in Figure 2.

2) IoT Deployments in a Residential Neighborhood: We next explore deployments of our 23 device types in a residential neighborhood from 2013 to 2017.

Input Datasets: We use DNS datasets from Case Connection Zone (CCZ) to study a residential neighborhood [2]. This dataset records DNS lookups made by around 100 residential houses in Cleveland, OH that connected to CCZ Fiber-To-The-Home experimental network and covers a random 7-day interval in each month between 2011 and 2017. Specifically, we apply DNS-based detection (both with and without server learning) to the January captures of 2013 to 2017 CCZ DNS data (Table II).

Results without Server Learning: As shown in Figure 6, from 2013 to 2017, we see roughly more detections and more types of device detected each year from this neighborhood. (Similar to §III-B1, to count per-device-type detection, we do not use detection merge.)

We believe our detection counts in Figure 6 lower-bound the actual IoT device counts in this neighborhood for two reasons: first, unlike our study on USC campus where dynamically assigned IPs inflate IoT detection counts (§III-A2), IPs in CCZ data are static to each house and do not cause such inflation; second, recalling that for NATted devices, our method only detects the existence of device types but cannot know the device counts for each type (§II-A), our detection counts in Figure 6 under-estimate IoT device counts if any household owns multiple devices of same types. We conclude that the lower bound of IoT device count in this neighborhood increases about 4 times from 2013 (at least 3 devices) to 2017 (at least 13 devices), consistent with our observation of increasing AS-level IoT deployment in this period.

We want to track IoT deployment by house but we can do that for only about half the houses because (according to author of this dataset) although IPs are almost static to

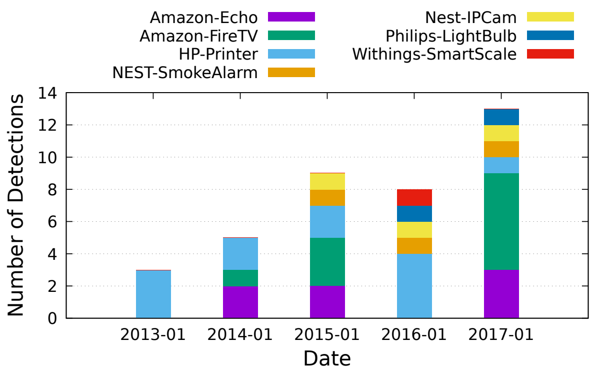

Fig. 6: IoT Deployments for All Houses in CCZ Data

2014-01	2015-01	2016-01	2017-01
HP_Printer	HP_Printer	HP_Printer	HP_Printer
	Nest_IPCam	Nest_IPCam	Nest_IPCam
	Nest_SmokeAlarm	Nest_SmokeAlarm	Nest_SmokeAlarm
		Philips_LightBulb	Philips_Bulb
		Withings_Scale	

TABLE V: IoT Deployment for One House in CCZ Data

each house, about half of the houses are rentals and see natural year-to-year variation from student tenants. Our detection results are consistent with this variation: most IPs with IoT detections at one year cannot be re-detected with the same set of device types in the following years.

We show the increasing IoT deployment can also be observed from a single house by tracking one house whose tenant looks very stable (since it is detected with consistent set of IoT device types over the 5 years). As shown in Table V, this household owns none of our known device types in 2013 (omitted in the table) and acquire HP_Printer in 2014, Nest_IPCam and Nest_SmokeAlarm in 2015, as well as Philips_LightBulb and Withings_SmartScale in 2016. Withings_SmartScale is missed in 2017 detection potentially due to this type of device generates no background traffic and it is not used during the 7-day measurement of 201701 CCZ data.

Results with Server Learning: With server learning, we see no additional detections. We do observe that during our detection to 5 years' CCZ DNS data, 951 distinct server names are learned and 3 known IoT device types are split. By analyzing these new server names, we conclude that server learning could discover new sub-types of known IoT device type but risk learning wrong servers from NATted traffic.

We first show server learning could learn new device server names and even new sub-type for known IoT device types. HP_Printer is originally mapped to 3 server names (per prior knowledge obtained in §III-A1). In the 2015-01 detection (others are similar), we learn 9 new server names for it in first iteration. But with these updated 12 server names, we find 2 less HP_Printer in subsequent detection, suggesting HP_Printer is in fact an aggregation of two sub-types (just like we merge Belkin_Switch and Belkin_MotionSensor as one type in §II-A1): one sub-type talk to the original 3 server names while the other talk to the updated 12 server names. We split HP_Printer into two sub-types and re-discover the two missed HP_Printer in subsequent detection.

We show our method risks learning wrong servers for a given IoT device type P behind NAT if there are non-IoT devices behind the same NAT visiting servers run by P's manufacturer. This is caused by two limitations in our method design: first, our method tries learning all unknown server names queried by IoT user IPs (§II-A3) because we cannot distinguish between DNS queries from detected IoT devices and DNS queries from other non-IoT devices behind the same NAT; second, we risk mis-classifying human-facing manufacturer server (that also serve non-IoT devices) as device server because we use necessary condition to determine device-facing server in §II-A1. In the 2015-01 detection (others are similar), we learn suspiciously high 176 device servers for Amazon_Echo and 277 device servers for Amazon_FireTV in first iteration, suggesting many of these new servers are learned from non-IoT devices (like laptops using Amazon services) behind the same NAT as the detected Amazon devices (because IoT devices usually only talk to at most 10 servers per day [38]). This false learning poisons our knowledge of device servers and causes us to detect two less Amazon_FireTV and one less Amazon_Echo in second iteration. Luckily, our method splits Amazon_Echo and Amazon_FireTV into two sub-types where one sub-type still mapped to the original, un-poisoned, set of device servers, allowing us to re-discover these missing Amazon devices in subsequent detections.

(We observe good performance in validation §IV-B where we apply server learning inside the NAT.)

C. Certificate-Based IoT Detection Results

Certificate-based detection only applies to devices that directly provide public web pages. IP cameras and Network Video Recorders (NVR) both often export their content, so we search for these. We find distinguishing them is hard because IP camera manufacturer often also produce NVR and to distinguish them requires finding non-manufacturer keys "IP Camera" and "NVR" in TLS certificates (per rules in §II-B1). Since we find certificates rarely contains these two text strings, we do not try to distinguish them and report them together as "IPCam".

Input Datasets: We apply detection to ZMap's 443-https-ssl_3-full_ipv4 TLS certificate dataset captured on 2017-07-26 [43] (as in Table II). This dataset consists of certificates found by ZMap TCP SYN scans on port 443 in the public IPv4 address space.

We target IPCam devices from 31 manufacturers (obtained from market reports [14], [15] and top Amazon sellers). We build matching keys for these IPCams based on rules in §II-B1.

Initial Detection Results: Table VI shows 244,058 IPCam devices we detect (represented by IoT certificates, 0.46% of all 52,968,272 input TLS certificates) from 9 manufacturers (29% of 31 input manufacturers, we do not see any detection from other 22 manufacturers). Among the detected devices, most (228,045, 93.43%) come from the top manufacturer Dahua. (Dahua is responsible for most IP cameras used in one DDoS attack [29].) Almost all (243,916, 99.94%) detected devices come from the top 5 manufacturers.

Partial Validation: Due to lack of ground truth, it is not possible to directly validate our results. We indirectly validate our results by accessing (via browser) IPs of 50 random candidate certificates from each IPCam manufacturers where we found at least one candidate certificate. If browser accessing shows a login screen with the correct manufacturer name on it, we consider it valid. This validation is limited since even a true positive may not pass it due to the device may be off-line or not show the manufacturer when we try it. (Our validation tests were done only 3 days after TLS certificate collection, to minimize IP address changes.)

Table VII shows our results, with 66% of detections correct. For the 106 false positives, in 40 cases the IP address did not respond and in 53 cases, we get login screen showing no manufacturer information. All 33 false negatives are due to Foscam IPCam fail our two rules to find IoT certificates in §II-B2: they are signed by a CA called "WoSign" and have uncommon C_{CN} place holder *.myfoscam.org.

By adding a special rule for Foscam devices (candidate certificates of Foscam that are signed by WoSign and have *.myfoscam.org as C_{CN} are IoT certificates), our detection correctness percentage increases to 70% (283 out of 404, with 15 true negatives becoming false positives due to we cannot confirm ground truth for 15 newly detected Foscam IPCam) and false negative percentage drops to 0%.

Revised Detection Results: Last row of §II-B shows our revised detection results with the special rule for Foscam: with 10,524 more detected Foscam devices, we have a total of 254,582 IPCam detections. (Our results show the subset of IPCams that are on the public Internet using TLS, but omit devices on private addresses and those not using TLS, as per §II-B.)

Geo-location Analysis: We geo-locate our revised detection result with Maxmind data published on 2017-07-18 (8 days before collection of the TLS certificate data we use) and find our detected IPCams come from 199 countries.

We examine what devices are in each country to gain confidence in what we detect. Table VIII shows the top ten countries by number of detected devices, and breaks down how many devices are found in country by manufacturer. (We show show only manufacturer with at least 1000 global detections in Table VI.)

We find manufacturers prefer different operating regions. We believe these preferences are related to their business strategies. While Dahua, Foscam and Hikvi-

Manufacturer	Dahua	Hikvision	Amcrest	Mobotix	Foscam	Vivotek	Tyco Intl	Schneider	NetGear	Axis Comm	Exacq	Arecont Vision	Apexis
Candidate Certificates	228,080	9,243	5,458	956	10,833	95	60	4	1	31	12	5	1
IoT Certificates	228,045	9,169	5,458	954	290	77	60	4	1	0	0	0	0
Adding Foscam Rule	228,045	9,169	5,458	954	10,814	77	60	4	1	0	0	0	0

TABLE VI: IPCam Detection Break-Down

Devices studied	404	(100%)		
Correctness	265	(66%)		
Incorrectness	139	(34%)	*(100%)*	
False Positives	106	(26%)	*(76%)*	*(100%)*
IP Non-Responsive	40	(10%)	*(29%)*	*(38%)*
Login w/o Mfr Info	53	(13%)	*(38%)*	*(50%)*
False Negatives	33	(8%)	*(24%)*	

TABLE VII: Partial Validation of Certificate-Based Detection Results

Country	Total	Dahua	Foscam	Hikvision	Amcrest	Mobotix
USA	47,690	38,139	3,666	655	5,038	143
S.Korea	22,821	22,520	84	212	4	0
India	19,244	19,029	23	186	6	0
China	17,575	15,539	288	1,748	0	0
Vietnam	14,092	13,794	113	176	9	0
France	8,006	7,059	506	372	1	62
Mexico	7,868	7,593	71	158	34	11
Poland	7,252	6,870	171	200	1	9
Argentina	6,384	6,141	154	75	13	0
Romania	5,646	5,272	139	207	2	23

TABLE VIII: Detected IP cameras and NVRs by Countries

sion are global,the latter two show substantially more deployment in the U.S. and China, respectively. Amcrest (formerly Foscam U.S. [7]) is almost exclusive to the American market. The German company Mobotix, while is present in Europe and America, seems completely absent from Asian markets.

IV. Validation

We validate the accuracy of our two main methods by controlled experiments.

Validation requires ground truth, so we turn to controlled experiments with devices we own. We have 10 devices (Table I) from 7 different manufacturers and at different prices (from $5 to $85, in 2018). This diversity provides a range of test subjects, but the requirement to own the devices means our controlled experiment is limited in size. In principle, we could scale up testing by by crowd-sourcing traffic captures, as shown in [20].

Our experiments also show our method correctly detects multiple devices from same manufacturer (3 devices from Amazon and 2 from TP-Link, as in Table I) using device merge and detection merge (recalling §II-A1).

A. Accuracy of IP-Based IoT Detection

We validate the correctness and completeness of our IP-based method by controlled experiments. We set up our experiment by placing our 10 IoT devices (Table I) and 15 non-IoT devices in a wireless LAN behind a home router. We assign static IPs to these 25 devices. We run tcpdump inside the wireless LAN to observe all traffic from the LAN to the Internet.

We run our experiments for 5 days to simulate 3 possible cases in real-world IoT measurements.

On Day 1 to 2 (*inactive days*), we do not interact with IoT devices at all. So first 2 days' data simulates observations of unused devices and contains only background traffic from the devices, not user-driven traffic. On day 3 to 4 (*active days*), we trigger the device-specific functionality of each of the 10 devices like viewing the cameras and purchasing items with Amazon_Button. The first 4 days' data shows extended device use. On day 5, we reboot each device, looking how a restart affects device traffic.

Our detection algorithm uses the same set of device server names that we describe in §III-A1. We collect IPv4 addresses for these device server names (by issuing DNS queries every 10 minutes) during the same 5-day period at the same location as our controlled experiments.

Detection During Inactive Days: We begin with detection using the first 2 days of data when the devices are inactive. We detect more than half of the devices (6 true positives out of 10 devices); we miss the remaining 4 devices: Amazon_Button, Foscam_IPCam, Amcrest_IPCam, and Amazon_Echo (4 false negative). We see no false positives. (All 15 no-IoT devices are detected as non-IoT.) This result shows that short measurements will miss some inactive devices, but background traffic from even unused devices is enough to detect more than half.

Detection During Inactive and Active Days: We next consider the first four days of data, including both inactive periods and active use of the devices. When observations include device interactions, we find all devices.

We also see one false positive: a laptop is falsely classified as Foscam_IPCam. We used the laptop to configure the device and change the device's dynamic DNS setting. As part of this configuration, the laptop contacts ddns.myfoscam.org, a device-facing server name. Since the Foscam_IPCam has only one device server name, this overlap is sufficient to detect the laptop as a camera. This example shows that IoT devices that use only a few device server names are liable to false positive.

Applying Detection to All Data: When we apply detection to the complete dataset, including inactivity, active use, and reboots, we see the same results as without reboots. We conclude that user device interactions is sufficient for IoT detection; we do not need to ensure observations last long enough to include reboots.

Simulating Dynamic IPs: We next show how dynamically assigned IPs can inflate IoT detections (both at USC, §III-A2 and at an IXP, §III-A3).

We simulate dynamic-assigned IPs by manually reassigning random static IPs to our 25 devices every day during our 5-day experiment.

Our IP-based detection with this simulated 5-day dynamic-IP measurements finds 26 true positive IoT detections from 25 dynamic IPs. One IP is detected with two IoT devices because they were each assigned to this IP on a different day. Similar to our 4-day and 5-day static-IP detection, we see a false detection of a laptop as Foscam_IPCam, and no false negatives. This experiment showed 2.6× more IoT devices than we have, less than the 5× inflation that would have occurred with each device being detected on a different IP each day.

We conclude that dynamic addresses can inflate device counts, and the degree depends on address lease times.

B. Accuracy of DNS-Based IoT Detections

We validate correctness and completeness of our DNS-based detection method by controlled experiments. We use the same set up, devices and device server names as in §IV-A. We also validate our claim that DNS-based detection can be applied to old network measurements by showing IoT device-type-to-server-name mappings are stable over time.

We run our experiments for 7 days and trigger device-specific functionality of each of the 10 devices every day to mitigate the effect of DNS caching.

We first apply detections with the complete set of device server names to evaluate the detection correctness and server learning performance of our DNS-based method. We then detect with incomplete set of device server names to test the resilience of detection and server learning to incomplete prior knowledge of device servers.

Detection with Complete Server Names: Results show 100% correctness (10 true positives and 15 true negatives), with 13 new device server names learned and 1 known device type splitted.

By analyzing the detection log, we show server learning and device splitting can correct incorrect device-merges. Recall in §III-A1, we merge TPLink_Plug and TPLink_LightBulb as one type (TPLink_Plug/Bulb) per our prior knowledge, they talk to the same server name devs.tplinkcloud.com. After first iteration of detection, we learn a new server deventry.tplinkcloud.com for TPLink_Plug/Bulb (from a detected TPLink_LightBulb, as shown by ground truth). However with now 2 server names mapped to TPLink_Plug/Bulb, we see one less detection of it in second iteration (ground truth shows a TPLink_Plug becomes un-detected). This reduced detection suggests TPLink_LightBulb and TPLink_Plug are in fact different device types: the former talks to the updated set of servers (devs.tplinkcloud.com and deventry.tplinkcloud.com) while the latter talk to the original set of servers (devs.tplinkcloud.com). We split TPLink_Plug/Bulb back into two to fix this incorrect device merge and re-discover the missed TPLink_Plug in subsequent detections.

Percentage of Mapping Dropped	Detection Correctness	Mapping Learned Back/Dropped	Learn-back Ratio
0%	100%	–	–
10%	100%	5/8	63%
20%	96%	6/15	40%
30%	96%	10/22	46%
40%	92%	11/29	38%
50%	96%	21/36	58%

TABLE IX: Resilience of Detection and Server Learning

Detection with Incomplete Set of Server Names: We detect with incomplete set of device server names to test resilience of detection and server learning to incomplete prior knowledge. Our goal is to simulate cases where we do not know all servers devices contact. We can have incomplete information should we not learn for long enough from them prior to detection (§II-A1), or because they change servers over time (perhaps due to firmware changes).

We randomly drop 10%, 20% to 50% known device-type-to-server-name mappings while ensuring each device type is still mapped to at least one server. We then compare the detection correctness and the learn-back ratio (how many dropped mappings are learned back after detections) of each experiment.

Results (Table IX) show our detection correctness are fairly stable: with 50 % servers dropped we still have 96% correctness. We believe two reasons cause this high correctness: our detection method suppress false positive (by ensuring device servers are not likely to serve human and IoT devices from other manufacturers) and the way we drop servers (ensuring each device mapped to at least one server name) guarantee low false negatives.

We also find the learn-back ratio is relatively stable, fluctuating around 50%. To explore how false detection happen and why about half dropped mappings cannot be learned back, we closely examine the detection and server learning with 20% (15) mappings dropped (others are similar). This experiment has only one false detection: Belkin_Plug is not detected due to 2 of its 3 server names are dropped while the remaining 1 server name is not queried in validation data. This experiment fail to learn back 9 of 15 dropped mappings: 4 due to server names not seen in validation data, 2 due to non-detection of Belkin_Plug (recall we only try to learn server from detected devices) and the rest 3 due to server names are not considered unknown (recall we only try to learn unknown servers) because they are originally mapped to both Amazon_FireTV and Amazon_Echo and we only dropped them from server list of Amazon_Echo.

Stability of Device Server Names: We support our claim that DNS-based detection can be applied to old network measurements by verifying IoT device-type-to-server-name mappings are stable over time. We show 8 of our 10 IoT devices (Table I) and a newly purchased Samsung_IPCam talk to almost identical set of device server names across 1 to 1.5 years. We exclude Amazon_Echo and Amazon_FireTV from this experiment because they talk to large number of device servers (previously measured 15 and 45) and it is hard to track all of them over time. We update these 9 devices to latest firmwares on May, 2018, measure latest servers name they talk to and compare these servers name with those we used in detection (measured on Oct 2016 for 1 device, on Dec, 2016 for 6 devices and on June 2017 for 2 devices). We found these 9 devices still talk to 17 of the 18 device server names we measured from them 1 to 1.5 years ago. The only difference is D-Link_IPCam who changes 1 of its 3 device server name from signal.mydlink.com to signal.auto.mydlink.com. A close inspection shows signal.auto.mydlink.com is CNAME of signal.mydlink.com, suggesting although D-Link_IPCam change the server names it queries (making it less detectable for our DNS-based method) , it still talk to the same set of actual servers (meaning our IP-based method is un-affected).

V. Related Work

Prior groups considered detection of IoT devices:

Heuristic-based traffic analysis: IoTScanner detects LAN-side devices by passive measurement within the LAN [37]. They intercept wireless signals such as WiFi packets and identify existence of IoT devices by packets' MAC addresses. While their work require LAN access and cannot generalize to Internet-wide detection, our three methods apply to whatever parts of the Internet that are visible in available network measurements, and are able to categorize device types.

Work from Georgia Institute of Technology detects existence of Mirai-infected IoT devices by watching for hosts doing Mirai-style scanning (probes with TCP sequence numbers equal to destination IP addresses) [3]. Their detection reveals existence of Mirai-specific IoT devices, but does not further characterize device types. In comparison, our three detection methods reveal both existence and type of IoT devices. Our IP and DNS-based method cover general IoT devices talking to device servers rather than just Mirai-infected devices.

Work from University of Maryland detects Hajime infected IoT devices by measuring the public distributed hash table (DHT) that Hajime use for C&C communication [19]. They characterize device types with Censys [9], but types for most of their devices remain unknown. In comparison, our three detection methods detect existence of known devices and always characterize their device types. Our IP and DNS-based methods cover general IoT devices talking to device servers rather than just those infected by Hajime.

Machine-learning-based traffic analysis: Work from Ben-Gurion University of the Negev (BGUN) detect IoT devices from LAN-side measurement by identifying their traffic flow statistics with machine learning (ML) models such as random forest and GBM [26], [27]. They use a wide range of features (over 300) extracted from network, transport and application layers, such as number of bytes and number of HTTP GET requests.

Similarly, work from the University of New South Wales (UNSW) characterizes the traffic statistics of 21 IoT devices such as packet rates and average packet sizes and briefly discusses detecting these devices from LAN-side by identifying their traffic statistics with ML model (random forest) [38].

Comparing to work from BGUN from UNSW, our work uses different features: packet exchanges with particular device servers and TLS certificate for IoT remote access rather than traffic statistics or traffic flow features. While they use LAN-side measurement where traffic from each device can be separated by IP or MAC addresses, our IP-based and DNS-based methods can work with aggregated traffic from outside the NAT and cover IoT devices both on public Internet and behind NAT. Not requiring LAN-side measurement also enables our IP-based and DNS-based methods to do Internet-wide detection. Our certificate-based method covers HTTPS-Accessible IoT devices on public Internet by crawling TLS certificates in IPv4 space.

Work from IBM transforms DNS names into embeddings, the numeric representations that capture the semantics of DNS names, and classify devices as either IoT or non-IoT based on embeddings of their DNS queries using ML model (multilayer perceptron) [24]. In comparison, our three methods not only detect existence of IoT devices, but also categorize their device types. While they rely on LAN-side measurement to aggregate DNS queries by device IPs, our three methods do not require measuring from inside the LAN.

IPv4 scanners: Shodan is a search engine that provides information (mainly service banners, the textual information describing services on a device, like certificates from HTTPS TLS Service) about Internet-connected devices on public IP (including IoT devices) [36]. Shodan actively crawls all IPv4 addresses on a small set of ports to detect devices by matching texts (like "IP camera") with service banners and other device-specific information.

Censys is similar to Shodan but they also support community maintained annotation logic that annotate manufacturer and model of Internet-connected devices by matching texts with banner information [9].

Compared to Shodan and Censys, our IP-based and DNS-based methods cover IoT devices using both public and private IP addresses, because we use passive measurements to look for signals that work with devices behind NATs. These two methods thus cover all IoT devices that exchanges packets with device servers during operation. Our certificate-based method, while also relying on TLS certificates crawled from IPv4 space, provides a better algorithm to match TLS certificates with IoT related text strings (with multiple techniques to improve matching accuracy) and ensures matched certificates come from HTTPS servers running in IoT devices.

Work from Concordia University infers compromised IoT devices by identifying the fraction of IoT devices detected by Shodan that send packets to allocated but un-used IPs monitored by CAIDA [40]. Their focus on compromised IoT devices is different from our focus on general IoT devices. Due to their reliance on Shodan data, they cover devices with public IP while our IP-based and DNS-based method cover devices on both public and private IP. We also report IoT deployment growth over a much longer period (6 years) than they do (6 days).

Northeastern University infers devices hosting invalid certificates (including IoT devices) by manually looking up model numbers in certificates and inspecting web pages hosted on certificates' IP addresses [5]. In comparison, our certificate-based method introduces an algorithm to map certificates to IoT devices and does not fully rely on manual inspection.

Work from University of Michigan detects industrial control systems (ICS) by scanning the IPv4 space with ICS-specific protocols and watching for positive responses [28]. Unlike from their focus on ICS-protocol-compliant devices and protocols, our approaches considers general IoT devices. Our approach also uses different measurements and signals for detection.

VI. Conclusion

To understand the security threats of IoT devices requires knowledge of their location, distribution and growth. To help provide these knowledge, we propose two methods that detect general IoT devices from passive network measurements (IPs in network flows and stub-to-recursive DNS queries) with the knowledge of their device servers. We also propose a third method to detect HTTPS-Accessible IoT devices from their TLS Certificates. We apply our methods to multiple real-world network measurements. Our IP-based algorithm reports detections from a university campus over 4 months and from traffic transiting an IXP over 10 days. Our DNS-based algorithm finds about 3.5× growth in AS penetration for 23 device types from 2013 to 2018 and modest increase in device type density in ASes detected with these device types. Our DNS-based method also confirms substantial growth in IoT deployments at household-level in a residential neighborhood. Our certificate-based algorithm find 254K IP camera and NVR from 199 countries around the world.

Acknowledgments

We thank Arunan Sivanathan at University of New South Wales for sharing their IoT device data with us [38]. We thank Paul Vixie for providing historical DNS data from Farsight [35]. We especially thank Mark Allman for sharing his CCZ DNS Transactions datasets [2] and help run our code on partially un-encrypted version of this dataset.

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-17-2-0280. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] G. Acar, N. Apthorpe, N. Feamster, D. Y. Huang, Frank, and A. Narayanan. IoT Inspector Project from Princeton University. https://iot-inspector.princeton.edu/.
[2] M. Allman. Case Connection Zone DNS Transactions, January 2018 (latest release). http://www.icir.org/mallman/data.html.
[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium*, 2017.
[4] CAIDA. Routeviews prefix to AS mappings dataset. https://www.caida.org/data/routing/routeviews-prefix2as.xml.
[5] T. Chung, Y. Liu, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measuring and applying invalid SSL certificates: the silent majority. In *Proceedings of the 2016 Internet Measurement Conference*, 2016.
[6] Cloudflare. What is an IXP. https://www.cloudflare.com/learning/cdn/glossary/internet-exchange-point-ixp/.
[7] Dahua. Important message from Foscam digital technologies regarding US sales and service. http://foscam.us/products.html/.
[8] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol. RFC 4346, Internet Request For Comments, 2006.
[9] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A search engine backed by Internet-wide scanning. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2015.
[10] Dyn. Analysis of October 21 attack. http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.
[11] K. Egevang and P. Francis. The IP network address translator (NAT). RFC 1631, Internet Request For Comments, 1994.
[12] Gartner. IoT installed base forcast. https://www.statista.com/statistics/370350/internet-of-things-installed-base-by-category/.
[13] B. Gleeson, A. Lin, J. Heinanen, T. Finland, G. Armitage, and A. Malis. A framework for IP based virtual private networks. RFC 2764, Internet Request For Comments, 2000.
[14] GlobalInfoResearch. IP cam market report. https://goo.gl/254g2M.
[15] GlobalInfoResearch. NVR market report. https://goo.gl/sxQRis.
[16] H. Guo and J. Heidemann. IoT traces from 10 devices we purchased. https://ant.isi.edu/datasets/iot/.
[17] H. Guo and J. Heidemann. Detecting IoT devices in the Internet (extended). Technical report, USC/ISI, 2018.
[18] H. Guo and J. Heidemann. IP-based IoT device detection. In *Proceedings of Workshop on IoT Security and Privacy*, 2018.
[19] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin. Measurement and analysis of Hajime, a peer-to-peer IoT botnet. In *Network and Distributed System Security Symposium*, 2019.
[20] D. Y. Huang, N. Apthorpe, G. Acar, F. Li, and N. Feamster. IoT inspector: Crowdsourcing labeled network traffic from smart home devices at scale, 2019.
[21] B. Krebs. Krebs hit with DDoS. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/.
[22] P. Krzyzanowski. Understanding autonomous systems. https://www.cs.rutgers.edu/~pxk/352/notes/autonomous_systems.html.
[23] J. Kurkowski. lib tldextract. https://pypi.python.org/pypi/tldextract.
[24] F. Le, M. Srivatsa, and D. Verma. Unearthing and exploiting latent semantics behind DNS domains for deep network traffic analysis. In *Workshop on AI for Internet of Things*, 2019.
[25] P. Loshin. Details emerging on Dyn DDoS attack. http://searchsecurity.techtarget.com/news/450401962/Details-emerging-on-Dyn-DNS-DDoS-attack-Mirai-IoT-botnet, 2016.
[26] 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 *Proceedings of SAC*, 2017.
[27] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici. Detection of unauthorized IoT devices using machine learning techniques, 2017.
[28] A. Mirian, Z. Ma, D. Adrian, M. Tischer, T. Chuenchujit, T. Yardley, R. Berthier, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey. An Internet-wide view of ICS devices. In *Annual Conference on Privacy, Security and Trust (PST)*, 2016.
[29] Motherboard. 1.5 million hijacked cameras make an unprecedented botnet. https://motherboard.vice.com/en_us/article/8q8dab/15-million-connected-cameras-ddos-botnet-brian-krebs.

[30] Mozilla. Public suffix list. https://www.publicsuffix.org/.
[31] M. Müller, G. C. M. Moura, R. de O. Schmidt, and J. Heidemann. Recursives in the wild: Engineering authoritative DNS servers. In *Proceedings of ACM Internet Measurement Conference*, 2017.
[32] No-IP. Domain names provided by No-IP. http://www.noip.com/support/faq/free-dynamic-dns-domains/.
[33] OVH. DDoS didn't break VAC. https://www.ovh.com/us/news/articles/a2367.the-ddos-that-didnt-break-the-camels-vac.
[34] SCIP. Belkin Wemo switch communications analysis. https://www.scip.ch/en/?labs.20160218.
[35] F. Security. Passive DNS historical Internet database: Farsight DNSDB. https://www.farsightsecurity.com/solutions/dnsdb/.
[36] Shodan. Shodan search engine front page. https://www.shodan.io/.
[37] S. Siby, R. R. Maiti, and N. O. Tippenhauer. IoTscanner: Detecting privacy threats in IoT neighborhoods. In *Workshop on IoT Privacy, Trust, and Security*, 2017.
[38] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Characterizing and classifying IoT traffic in smart cities and campuses. In *Workshop on Smart Cities and Urban Computing*, 2017.
[39] ThousandEyes. What is an ISP? https://www.thousandeyes.com/learning/glossary/isp-internet-service-provider.
[40] S. Torabi, E. Bou-Harb, C. Assi, M. Galluscio, A. Boukhtouta, and M. Debbabi. Inferring, characterizing, and investigating Internet-scale malicious IoT device activities: A network telescope perspective. In *Conference on Dependable Systems and Networks*, 2018.
[41] USC/LANDER. FRGP (www.frgp.net) Continuous Flow Dataset, traces taken 2015-05-10 to 2015-05-19. provided by the USC/LANDER project (http://www.isi.edu/ant/lander).
[42] Wikipedia. Autonomous system (internet). https://en.wikipedia.org/wiki/Autonomous_system_(Internet).
[43] ZMap. ZMap 443 HTTPS SSL full IPv4 datasets. https://censys.io/data/443-https-ssl_3-full_ipv4.

Hang Guo Hang Guo received his B.S. degree from Beijing University of Posts and Telecommunications in 2014 and his Ph.D. degree from University of Southern California in 2020. His research interests include Internet traffic analysis, network security, and Internet-of-Things (IoT). In 2020, He joined the Microsoft Azure team as a software engineer.

John Heidemann John Heidemann (S'90, M'95, SM'04, F'14) received his B.S. from University of Nebraska-Lincoln (1989) and his M.S. and Ph.D. from the University of California, Los Angeles (1991 and 1995). He is a principal scientist at the University of Southern California/Information Sciences Institute (USC/ISI) and a research professor at USC in Computer Science. At ISI he leads the ANT (Analysis of Network Traffic) Lab, observing and analyzing Internet topology and traffic to improve network reliability, security, protocols, and critical services. He is a senior member of ACM and fellow of IEEE.