Detecting IoT Devices in the Internet (Extended)

USC/ISI Technical Report ISI-TR-726 July 2018

Hang Guo
USC/Information Sciences Institute
hangguo@isi.edu

John Heidemann
USC/Information Sciences Institute
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. We have developed these approaches with 10 device models from 7 vendors. Our third method uses TLS certificates obtained by active scanning. We have applied our algorithms to a number of observations. Our IP-based algorithms see at least 35 IoT devices on a college campus, and 122 IoT devices in customers of a regional IXP. We apply our DNS-based algorithm to traffic from 5 root DNS servers from 2013 to 2018, finding huge growth (about 7×) in ISP-level deployment of 26 device types. DNS also shows similar 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.

1. INTRODUCTION

There is huge growth in sales and the installed base of Internet-of-Things (IoT) devices like Internet-connected cameras, light-bulbs, TVs. Gartner forecasts the global IoT installed base will grow from 3.81 billion in 2014 to 20.41 billion in 2020 [7].

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 DDoS attacks that set records in attack bitrates. Estimated attack sizes include a 620 Gb/s attack against cybersecurity blog `KrebsOnSecurity.com` (2016-09-20) [12], 1 Tb/s-size attacks French cloud-computing provider OVH (2016-09-23) [19] and DNS provider Dyn (2016-10-21) [6], and 1.7 Tb/s in 2018 [15]. The size of the Mirai botnet used in these attacks has been estimated at 145k [19] and 100k [6]. Source code to the botnet was released [14], 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 these knowledge could help guide the development of future IoT security solutions.

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 §2.1.2 and stub-to-recursive DNS queries §2.1.3). 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 certificates they use, with minimal information from target devices: manufacturer names and some public product information (§2.2).

Our second contribution is to apply our three detection methods to multiple real-world network measurements. We apply our IP-based method to flow-level traffic from a college campus (§3.1.2) and a regional IXP (§3.1.3), revealing at least 35 IoT devices on campus and 122 devices in customers of the IXP. We apply our DNS-based method to DNS traffic at five root name servers from 2013 to 2018 (§3.2.1) and find about 7× growth in ISP-level IoT deployment for 26 device types. We confirm similar deployment growth at household-level by applying DNS-based method to DNS traffic from a residential

neighborhood from 2013 to 2017 (§3.2.2). We apply our certificate-based method to a public TLS certificate dataset (§3.3) 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 [24]. Others are currently studying the privacy and vulnerabilities of individual devices (for example [1]); we focus on detection. Prior work has studied detection [23, 24, 22, 5], but we use different detection signals to observe devices behind NATs as well as those on public IP addresses. We published an early version of IP-based detection in a workshop [11]. This paper adds two new detection methods: DNS-based detection (§2.1.3) and certificate-based detection(§2.2), extends analysis of IP-based detection from a half-day to 10 days.

Our algorithms and results can guide the design and deployment of future IoT security solutions by revealing the scale of IoT security problem (how many IoT devices are there? what is the worst possible IoT-based DDOS attack?), the problem's growth (how many IoT devices will there be in 5 years?) and the distribution of the problem (where are IoT devices? who are the potential customers of IoT security solutions?).

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 1) public for the scientific community at [10].

2. METHODOLOGY

We next describe our three methods to find IoT devices: two using traffic (§2.1), and the third, TLS certificates (§2.2).

2.1 IP and DNS-Based Detection Methods

Our two main methods detect general IoT devices both with public IPs and behind NAT. Our methods follow the insight that most IoT devices exchange traffic regularly with device-specific servers. 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 only consider whom IoT devices talk with but not the patterns of their talking like timing and rates because timing is often obscured when NATs mix traffic from multiple devices.)

These two methods therefore depend on identifying servers to look for (§2.1.1) and looking for these servers by IP address (§2.1.2) and DNS name (§2.1.3).

2.1.1 *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 shared across multiple types of devices, since they would otherwise produce false detections.

Identify 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 1 and provide the information we learned as a public dataset [10].

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 devices. The other is *human-facing servers*: servers that also serve human.

Third-party servers usually offer public services like time, news, 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 [17]). We use Python library tldextract to identify TLD suffixes [13].

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. This test is supported by the observation that retrieving HTML pages via HTTP or HTTPS is the most common method in which average users access web servers; and consuming web content is the most common purpose why average users access web servers.

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 1).

We call the remaining server domain 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.

We propose a technique to automatically discover new device server names during detection in our DNS-based method §2.1.3. (This technique cannot be replicated in IP-based method because it depends on server names in DNS queries.)

Handling Shared Server Names: Some device server names are shared among multiple IoT device types 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 merging*: 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.)

2.1.2 IP-Based IoT Detection Method

Our first method detects IoT devices by identifying packet exchanges between IoT devices and device servers. For each specific type of device, we track *device-to-server-name mapping*: a list of device server names that device 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 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 our test 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. (In §3.1.3, we show 58 of our 99 device server names change IP at least once in a 2-month period.) We also assume server-name-to-IP mappings could be location-dependent. (We show simultaneous DNS resolutions for 40 of our 99 device server names from §3.1.1 give different results in different geo-locations).

We track changes of server-name-to-IP mapping by resolving server names to IP addresses every hour. 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 §2.1.1 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.

Most devices talk to a handful of device server names (up to 20, from our laboratory measurements §3.1.1). For these types of devices, we require seeing at least 2/3 device server names to believe an 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. (We do not consider devices with 1 to 2 device servers names here because for these devices, any thresholds larger than 1/2 are effectively requiring all 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 (Table 1) has 41 device server names. Individual devices will talk to multiple device server names, but most likely only a subset of the pool, at least over short observations.

2.1.3 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 two limitations in IP-based detection: First, while server DNS

names are stable, server IP can change. Consequently, applying detection to old network measurement requires old IP addresses for device servers. Second, with DNS queries, we can discover new device server names by examining unknown server names queried by detected IoT devices and learning those look like device servers (using rules in §2.1.1). Server learning addresses the problem that our prior knowledge of device servers are potentially incomplete. (We cannot replicate this process in IP-based detection because we find it hard to judge if a unknown IP is device server, even with help of reverse DNS and TLS Certificate from that IP.)

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 resolvers. This method also works with recursive-to-authority DNS queries (see §3.2) when observations last longer than DNS records are cached, 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 ISP level (considering recursive servers are usually run by ISPs 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 conduct 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 could be learned. This method also revises newly learned server names by device splitting if it suspects they are false knowledge (signaled by decreased detections after learning new server names).

Detection: Similar to §2.1.2, for each type of IoT device, we track a list of device server names that device talks to. We interpret presence of DNS queries for above a threshold (same as §2.1.2) amount of device server names from a give IP address as presence of that IoT device. (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 §2.1.1). For example,"command-2" is the sub-domain of `command-2.amcrestcloud.com` and in detection, we consider `command-2.amcrestcloud.com` and `command-9.amcrestcloud.com` the same.

Server Learning: We learn new device server names from detected IoT devices and use them in subsequent detections. After each detection, we examined all unknown server names DNS queried by detected IoT user IPs. If a unknown server name S queried by an IoT user IP I looks like a device server (judged by rules in §2.1.1) for IoT device P detected at I, we add S to the list of device server names we track for P.

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

False device merge can cause reduced detection. When we falsely merge different device types $P1$ and $P2$ as P, we risk learning server names for P (from detected $P1$ or $P2$) that only apply for $P1$ or $P2$ and causing reduced detections of P in subsequent iterations.

Device splitting addresses this problem by reverting false merge. If we detect less P at certain IP after learning new device server names, we know P is in fact an aggregation of two different type of devices: one talk to server names mapped to P before last server learning; the other talk to the latest server names mapped to P. We split P into two following this new knowledge.

2.2 Certificate-Based IoT Detection Method

Our third method detects IoT devices directly connected to the public internet using HTTPS by identifying their TLS Certificates. There are other works that map TLS certificate to IoT devices either by matching texts (like "IP camera") with certificates [22] or by using community maintained annotation logic to do this certificate matching [5]. In comparison, our method not only uses multiple techniques to improve the accuracy of certificate matching, but it also confirms that matched certificates come from HTTPS servers running in IoT devices.

We use existing public crawls of the IPv4 TLS certificates. We first identify *candidate certificates*: the TLS certificates that contain 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 IoT devices. Each IoT certificate represents a HTTPS-Accessible IoT device.

2.2.1 *Identify Candidate Certificates*

We identify candidate certificates for a HTTPS-Accessible IoT device by testing each TLS certificate against a set of text strings we associate with this device (called *matching keys*).

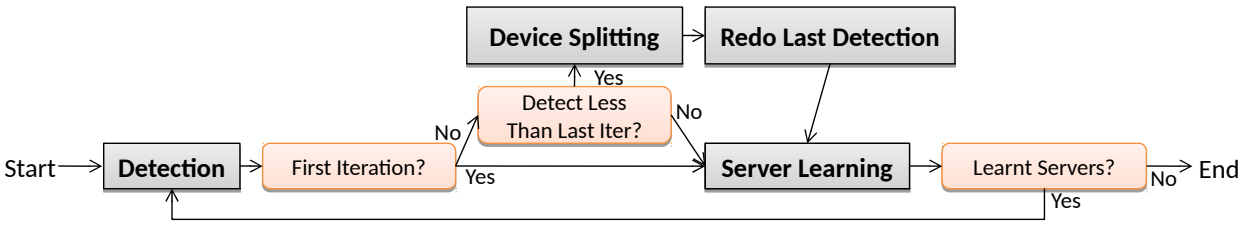

Figure 1: Work Flow for DNS-Based IoT Detection

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, 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 brands, even when they are manufactured by a larger organization. For example, "American Dynamics" is the 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 are 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".

Requiring a Good-Match for the manufacturer key reduces false positives from IoT manufacturer names that are substrings of other companies. For example, name of IP camera manufacturer "Axis_Communications" is a substring of third party organization "Maxis_Communications_Berhad".

We require Match for all other keys (product types and sub-brand) to be flexible and reduce false negatives. For example, non-manufacturer key "NVR" can be matched to text string like "myNVR".

Key Matching Algorithm: We examine each input TLS certificate (more specifically, their organization C_O , organization units C_{OU}, common name C_{CN} and optional SubjectAltNames C_{DN} fields) with matching keys from each target HTTPS-Accessible IoT device. If for a certificate C and a target device P, P's manufacturer key K_m is a Good-Match of C_O and P's non-manufacturer keys K_n are Match of any of C_O, C_{OU},C_{CN} or C_{DN}, we consider C a candidate certificate for P.

We handle two special cases in matching K_m with C_O. If C_O is empty or an apparent place holder like "SomeOrganization" and "company", we instead test if K_m is a Good-Match any of C_{OU}, C_{CN} or C_{DN}. If any of these four fields is URL, we only match K_m against the URL's domain part (URL's domain as defined in §2.1.1) because domain shows ownership of a server name. (For example, Accedo Broadband instead of Sharp owns `*.sharp.accedo.tv''`.)

2.2.2 *Identify IoT Certificate*

Because IoT devices are often self-generated, they are often not signed by Certificate Authorities (CAs). We use this information to detect them, requiring IoT certificates to be self-signed and not have a validate domain name.

Self Signing: We believe HTTPS servers running in IoT devices should rarely use CA-signed certificates because it is unlikely for average IoT users to acquire TLS certificates from CA for their IoT devices. We consider a certificate self signed if the certificate's issuer organization C_{iO} either equals to of any of C_O, C_{OU} and C_{CN} or contains K_m.

No 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 certificate to lack a valid domain name if none of the values in C_{CN} is a valid domain name. We do not count DDNS domain names (based on published DDNS domains from no-IP.com [18]) and apparent place holder as valid domain name.

3. RESULTS: IOT DEVICES IN THE WILD

We next apply our detection methods to observations of real-world network traffic to learn about the distribution and growth of IoT devices in the wild. (We later verify the accuracy of our approaches in §4.)

3.1 IP-Based IoT Detection Results

To apply our IP-based detection, we first extract device server names from 26 devices by 15 vendors (§3.1.1). We then apply detection to Internet flows at a college campus (§3.1.2) and from an IXP (§3.1.3).

Manufacturer	Model	Alias
Amazon	Dash Button	Amazon_DashButton
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_SmartPlug
TP-Link	HS100 Smart Plug	TPLink_SmartPlug
Philips (Hue)	A19 Starter Kit	Philips_LightBulb
TP-Link	LB110 Light Bulb	TPLink_LightBulb

Table 1: The 10 IoT devices that we purchased.

Manufacturer	Model	Alias
Amazon	Echo	Amazon_Echo
Belkin (Wemo)	Switch	Belkin_Switch
Belkin (Wemo)	Motion Sensor	Belkin_MotionSensor
Blipcare	Blood Pressure Meter	Blipcare_BPMeter
HP	Wireless Printer	HP_Printer
iHome	Smart Plug	iHome_SmartPlug
Insteon	IP Camera	Insteon_IPCam
Invoxia (Triby)	Smart Speaker	Invoxia_SmartSpeaker
LiFX	Smart Light Bulb	Lifx_LightBulb
Nest	Dropcam IP Camera	Nest_IPCam
Nest	Protect Smoke Alarm	Nest_SmokeAlarm
Netatmo	Weather Station	Netatmo_WeatherStation
Netatmo	Welcome IP Camera	Netatmo_IPCam
PIX-STAR	Wifi Photo Frame	PIX-STAR_PhotoFrame
Samsung	SmartCam HD Pro	Samsung_IPCam
Samsung	SmartThing Hub	Samsung_Hub
TPLink	Day Night Cloud Cam	TPLink_IPCam
TPLink	Smart Plug	TPLink_SmartPlug
Withings	Aura Sleep Sensor	Withings_SleepSensor
Withings	Smart Baby Monitor	Withings_BabyMoniter
Withings	Smart Scale	Withings_SmartScale

Table 2: The 21 IoT devices from UNSW

3.1.1 Identifying Device Server Names

We use device servers from two sets of IoT devices in detection: 10 IoT devices we purchased (Table 1) and 21 IoT devices from data provided by the University of New South Wales (Table 2, derived from Figure.1b of [24]) . We extract device server names from both sets of devices with method described in §2.1.1.

We show the count of server names we find from our 10 devices in Table 3. Of the 171 candidate server names from our 10 devices, about half (56%, 96) are third-party servers, providing time, news, or music streaming. As for the 75 manufacturer servers, only a small portion (7%, 5) of them are human-facing (like `prime.amazon.com`) while most (93%, 70) of them are device-facing manufacturer servers that will be used in detection. We also find two devices (TPLink_SmartPlug and TPLink_LightBulb) sharing their only server name (`devs.tplinkcloud.com`) and merge them to one meta-device for detection.

Candidate Server Names	171	(100.00%)	
3rd-Party Servers	96	(56%)	
Manufacturer Servers	75	(44%)	(100.00%)
Human-Facing Mfr Servers	5	(3%)	*(7%)*
Device-Facing Mfr Servers	70	(41%)	*(93%)*

Table 3: Server Extraction for Our 10 Devices

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. We list them as third-party servers and do not use them in detection. We find 3 server candidate names (`api.xbcs.net`, `heartbeat.lswf.net`, and `nat.xbcs.net`) falsely classified as third-party server. We confirm they are run by IoT manufacturer Belkin from query result of "whois `lswf.net`" and a study from security company SCIP [20] and add them back to our list. These three server names fail our test for manufacturer server (§2.1.1) because their domains show no information of manufacturer.

Similarly, we extracted 48 device server names from 18 of 21 IoT devices from University of New South Wales (using datasets available on their website http://149.171.189.1/). The remaining 3 of their IoT devices are not detectable with our method because they only visit third-party servers and human-facing manufacturer servers.

Combining server names measured from our 10 devices and the 18 detectable devices from University of New South Wales (merging 2 duplicated devices: Amazon_Echo and TPLink_SmartPlug) gives us 26 types of IoT devices; together they have 99 distinct device server names.

3.1.2 IoT Devices in a College Campus

To test our IP-based detection method, we apply it to network traffic from part of our university campus.

Input Datasets: We use passive Internet measurements at the University of Southern California (USC) guest WiFi from 2017-10-06 to 2017-10-11 (6 days). 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 be dynamic across time and geolocation (as discussed in §2.1.2), to make sure we get the correct bindings, we collect IPv4 addresses for our 99 device server name near USC, across the same 6-day period, as described in §2.1.1.

IP A & B & C	IP D	IP E	IP F	IP G & H
Belkin_SmartPlug	Samsung_IPCam	Samsung_IPCam	HP_Printer	Amazon_*
Samsung_IPCam	HP_Printer	HP_Printer	Withings_Scale	
HP_Printer	LiFX_LightBulb	LiFX_LightBulb	Amazon_*	
Netatmo_WeatherStation	Amazon_*	Amazon_*		
LiFX_LightBulb	Withings_*			
Amazon_*				
Withings_*				

Table 4: IoT Devices In USC Campus

Detection Results: We see a total of 35 triggered detections (suggesting at least 35 detected devices) from 8 user IPs, as shown in Table 4. Note that "Amazon_*" in Table 4 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 §2.1.1).

Our first observation is IPs with IoT devices often have several devices, suggesting the use of a network-address translation (NAT) device that uses a single IP address from USC's WiFi.

We also find three IoT user IPs (IP A, B, and C) sharing the exact set of IoT devices. A likely explanation is that there is one user with a set of devices behind NAT that is using a dynamically assigned IP address, and that this address changes three times over our six day study. We confirm these three IPs belong to three different users by verifying these three IPs are seen at nearly the same time in the network traffic.

These observations show our approach works at a real campus. However, our measurements at USC represent only a small fraction of campus network traffic—we see only guest network WiFi, not wired networks and secure WiFi. These results therefore represent a lower bound for actual IoT deployment at USC campus.

3.1.3 IoT Devices at an IXP

We also apply IP-based detection to partial traffic from an IXP.

Input Datasets: We use the FRGPContinuousFlowData dataset [26], abbreviated as *FRGP*, collected by Colorado State University (CSU) from 2015-05-10 to 2015-05-19 (10 days). This dataset consists of anonymized Internet traffic flows in Argus format from the Front-Range Gigapop (www.frgp.net) connecting customers of that regional network (including 18 universities and 14 large organizations in Colorado) with two commercial ISPs: Century Link and Comcast. Data is provided as Argus-format flow records, with anonymized IP addresses.

Input Server IPs: We do not know IPs for our device servers in 2015. So we draw upon IPv4 addresses we collect for our 99 device server names from 2017-10-12 to 2018-2-23 near USC (with method in §2.1.1) and show IPs for our 99 device server names are either stable over time or from stable pools.

We have considered the commercial historical DNS dataset from Farsight Security [21] but we find this dataset has very limited coverage: Farsight data collected from an extended two-year period only covers IPs for 51 of our 99 device server names, giving us at most 11 detectable IoT device types. We show detection results (60 detections of only 2 type of devices) with 2-year Farsight DNS data in our workshop paper [11].

We verify our server IPs collected 2 years after FRGP data are still applicable by confirming IPs for our 99 device servers are either stable over time or rotating within stable pools. We collect server IP history for our 99 device server names from 2017-10-20 to 2017-12-28 and confirm that none of them really changes IP in this 2-month period: more than half of them (58 out of 99) rotate IPs within pools of IPs while the rest (41) keep using the same IP mappings.

Since our collection of server IPs (at USC) does not co-locate with collection of FRGP data (at CSU), any location-dependent IPs we collect will not be applicable to FRGP data.

Detection Results: Our detection results show 122 triggered detections (suggesting at least 122 detected devices) of at least 9 device types (we do not know exact number of types due to detection merge and device merge §2.1.1) from 111 IPs.

We conclude that with our IP-based detection method, it is hard to detect IoT devices in the past because server IPs change over time (and potentially across geo-location) and commercial historical DNS dataset has limited coverage.

3.2 DNS-Based IoT Detection Results

Root	201305	201404	201504	201604	201610	201704	201709	201804
B	0	2	5	39	109	*519*	24	20
A	0	4	6	78	—	21	—	39
C	0	2	5	41	—	268	—	44
K	0	3	3	77	—	42	—	9
I	0	4	4	60	—	59	—	21

Table 5: Samsung_IPCam Detection Comparison

We next apply our DNS-based detections with our 26 IoT device types to two real-world DNS datasets.

3.2.1 *Global ISP-Level IoT Deployments*

We apply detection to Day-in-the-Life of the Internet (DITL) datasets from 2013 to 2018 to explore growth of ISP-level deployments of our 26 device types.

Input Datasets: Our detection uses DITL datasets from five root DNS servers (A,B,C,K and I) between 2013 and 2018 (data from DNS-OARC [4] and USC/ISI [25]). Each DITL data contains DNS queries received by a root DNS server in a 2-day window. We mainly show detection results of B root because it has two more data points (2016-10-04 and 2017-09-19)

Since root DNS servers see requests from recursive DNS servers (usually run by ISPs for their users), these results detect devices at the ISP-level, not for households.

Since the data represents ISPs and instead of households, we do detection only (§2.1.3) and omit the server-learning portion of our algorithm. With many households mixed together, ISP-size aggregation risk learning wrong servers. To count per-device detections, we do not use detection merge (§2.1.1).

Trend in ISP-Level Deployment: The overall detection results for B root (Figure 2) and other 4 roots (Figure 4) show increases in both number of detection and types of device detected from 2013-05-28 to 2017-04-11, suggesting an increasing ISP-level deployments for our 26 device types in this period.

The overall number of detection for A, C, K and I roots (Figure 4) plateau after 2017-04-11, suggesting the ISP-level deployments for our 26 IoT device types stop increasing after 2017.

The overall number of detection for B root (Figure 2), however, seems start declining rather than plateauing after 2017-04-11 DITL.

To understand the cause of this drop in B root's overall detections, we inspect B-root results by device. As shown in Figure 3, we find the drop in B root's overall detections from 2017-04-11 to 2017-09-19 is mostly due to reduced detection of Samsung_IPCam (from 519 to 24). We confirm software updates are not the cause for this reduced detection by purchasing a Samsung_IPcam and confirm its latest firmware (as of April, 2018) still talk to the same servers we used in detection (measured on Oct, 2016). Another possible explanation is sampling error: large amount of DNS queries from Samsung_IPCam happen to be directed to B root on 2017-04-11, causing an excessively high detection of Samsung_IPCam in 2017-04-11 B-root DITL data and a "reduced" detection in 2017-09-19 data. We prove this hypothesis by comparing detections of Samsung IPCam across 5 roots (Table 5) and showing the 519 Samsung_IPCam detection in 2017-04-11 B-root data is indeed excessively high comparing to other 4 roots' detections on 2017-04-11 and all 5 roots' detections after 2017-04-11.

If we set aside this excessively high detection of Samsung_IPCam in 2017-04-11, we get a clearer picture of ISP-level IoT deployment shown by B-root detection results (Figure 2): deployments of our 26 device types keep increasing from 2013-05-28 to around 2017-04-11 and plateau afterward, agreeing with the trends shown by other 4 roots (Figure 4).

We conclude that from 2013-05-28 to 2017-04-11, ISP-level deployment of our 26 device types has increased about 7× (averaged over detection results of A,C,K and I due to B-root detection is excessively high on 2017-04-11)

Causes Of the Trend: We believe the reason overall deployment of our 26 IoT device types stop increasing after 2017-04-11 is that most of them are released a couple years before 2017 and their sales have been slowing down over the years.

We show estimated release dates for our IoT device types are consistent with this hypothesis. We estimate release dates for 25 of our 26 IoT device types (except HP_Printer from University of New South Wales, where there are many possible models) from one of three sources (ordered by priority high to low): release date found online, device's first appearance date on US Amazon and device's first customer comment date on US Amazon. (When there are multiple models for one IoT device type, we estimate release dates for all models and use the earliest date for this device type.) We confirm most (23 out of 25) device types are indeed released at least two years before 2017: 2 in 2011, 7 in 2012, 4 in 2013, 5 in 2014 and 5 in 2015.

We also compare these estimated release dates with B-root per-device results (Figure 3) and find device types released early tend to flat out in detection early, consistent with product cycles and a decrease in sales and use of these devices.. For example, Withings_SmartScale and Netatmo_WeatherStation,

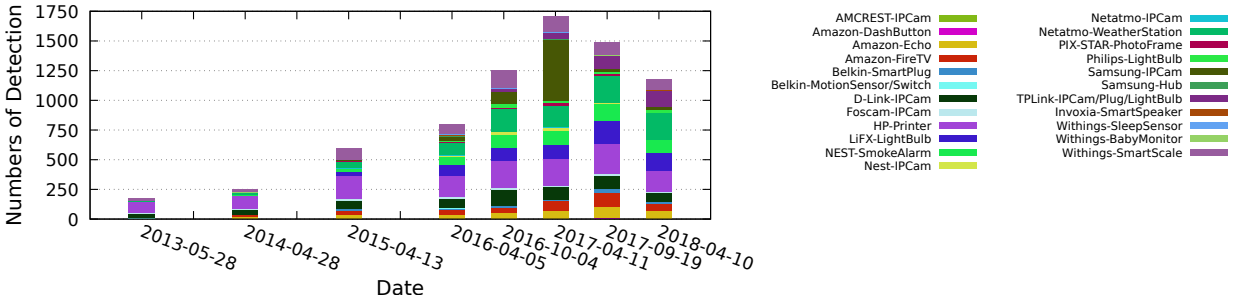

Figure 2: B-Root Overall Detection Results for All 26 IoT Device Types

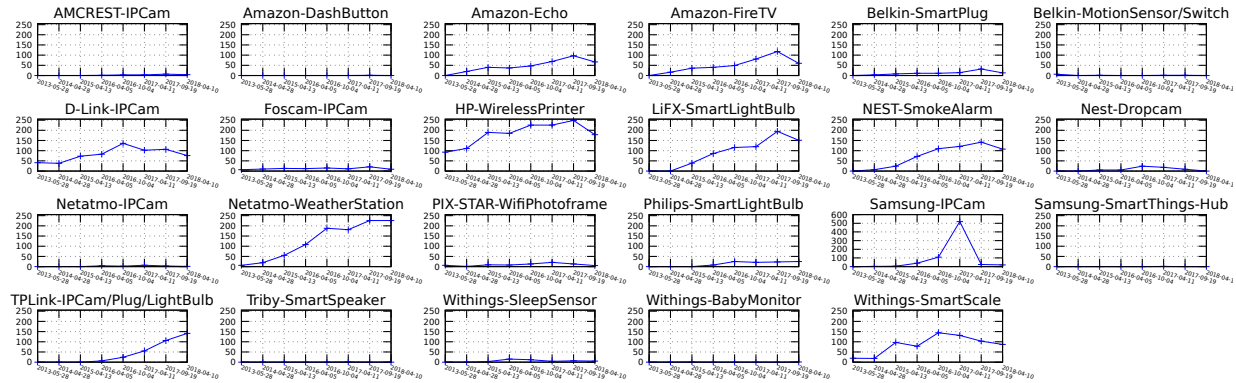

Figure 3: B-Root Per-Device Detection Results

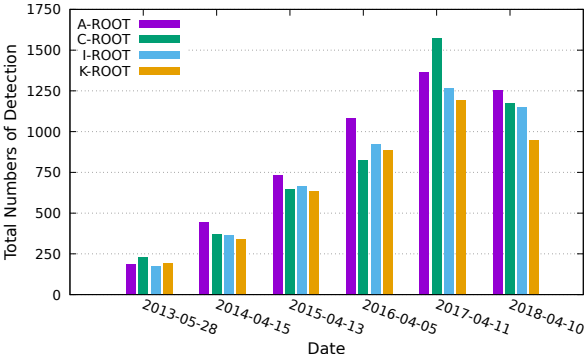

Figure 4: A,C,K and I Roots' Overall Detection Results for All 26 IoT Device Types

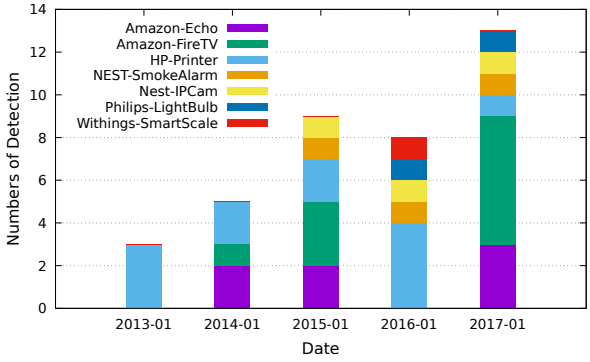

Figure 5: IoT Deployment of All Approximately 100 Houses in CCZ DNS Data

released in 2012, stop increasing roughly after 2016-10-04, suggesting a product cycle of about 4 years. In comparison, TPLink-IPCam/Plug/LightBulb are the small portion of device types released around 2016 (TPLink_IPCam on 2015-12-15, TPLink_SmartPlug on 2016-01-01 and TPLink_Lightbulb on 2016-08-09) and their deployments continue to rise even on 2018-04-10, despite deployments of most other device types (released between 2013 and 2015) roughly stop increasing by 2017.

We rule out the possibility that the increasing detection we observe from 2013 to 2017 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-to-server-name mappings are stable over time in §4.2.

3.2.2 *IoT Deployments in a Residential Neighborhood*

We next explore deployments of our 26 device

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_LightBulb
		Withings_Scale	

Table 6: IoT Deployment of One House in CCZ Data

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 internal 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.

Results without Server Learning: As shown in Figure 5, from 2013 to 2017, we see roughly more detections and more types of device detected each year from this neighborhood, agreeing with our observation of increasing ISP-level IoT deployment in this period (§3.2.1). (As in §3.2.1, to count per-device detection, we do not use detection merge.)

We want to track IoT deployment by household but we can do that for only about half the houses because (according to author of this dataset) although IPs are almost static to each house, approximately 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 devices in the following years.

We show the increasing IoT deployment can also be observed from a single household by tracking one house whose tenant looks very stable (since it is detected with consistent set of IoT devices over the 5 years). As shown in Table 6, this household owns none of our 26 types of devices 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; this omission may be because this device generates no background traffic (potentially to save battery) and it is not used during the 7-day measurement of 201701 CCZ DNS data.

Results with Server Learning: With server learning, we see no additional detections. We do observe overall 951 distinct server names are learned and 3 known IoT device types are split during detection to 5 years' CCZ DNS data. By analyzing these new server names, we conclude server learning could discover new sub-types of known IoT device type but risk learning wrong servers from NATed 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 §3.1.1). 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 TPLink_SmartPlug and TPLink_SmartPlug as one type in §2.1.1): 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 and re-discover the two missed HP_Printer in subsequent detection.

We show our method risks learning wrong servers for a given NATed IoT device P 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 IP (§2.1.3) 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 §2.1.1.

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 detected Amazon devices because IoT devices usually only talk to at most 10 servers per day (as shown by [24]). This false learning poisons our knowledge of device servers and causes we 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 §4.2 where we apply server learning inside the NAT.)

3.3 Certificate-Based IoT Detection Results

Certificate-based IoT detection only applies to devices that directly provide public web pages. IP

Figure 6: Dahua IPCam found at 14.164.62.67

cameras and Network Video Recorders (NVR) both often export their content, so we search for these. We find distinguishing between 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(recall rules in §2.2.1). (We also find TLS certificates rarely contains these two text strings.) Therefore we do not try to distinguish them and report them together as "IPCam". (Since many IP-Cams' manufacturers only produces IPCam, we can find their candidate certificates by simply searching manufacturer names.)

Input Datasets: We apply detection to ZMap's 443-https-ssl_3-full_ipv4 TLS certificate dataset captured on 2017-07-26 [27]. 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 [8, 9] and top Amazon sellers). We build matching keys for these IPCams based on rules in §2.2.1.

Initial Detection Results: Table 7 shows the 244,058 IPCam devices we find (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 (who are believed to be responsible for most compromised IP cameras in DDoS attack against `KrebsOnSecurity.com` [16]). Almost all (243,916, 99.94%) detected devices come from the top 5 manufacturers.

Partial Validation: Due to lack of groundtruth, it is not possible to directly validate our detection 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. Figure 6 shows an example of a login screen we found with logo and name of IPCam manufacturer Dahua.

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 only 3 days after TLS certificate collection, to minimize IP address changes. prevent possible network change.

Table 8 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 §2.2.2: 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 become false positives due to we cannot confirm groundtruth for 15 newly detected Foscam IPCam) and false negative percentage drops to 0%.

Revised Detection Results: Last row of §2.2 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.

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 break down detection results by country in A.) .

4. VALIDATION

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

4.1 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 1) and 15 non-IoT devices in a wireless LAN behind a home router. 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

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 7: IPCam Detection Break-Down

Devices studied	404	(100%)		
Correctness	265	(66%)	*(100%)*	
True Positives	191	(47%)	*(72%)*	
True Negatives	74	(18%)	*(28%)*	
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 8: Partial Validation of Certificate-Based Detection Results

All Device Present	25	(100%)	
Correctness	24	(96%)	*(100%)*
True Positive	10	(40%)	*(41.67%)*
True Negative	14	(56%)	*(58.33%)*
Incorrectness	1	(4%)	*(100%)*
False Positive	1	(4%)	*(100%)*
False Negative	0	(0%)	*(0%)*

Table 9: Detection Result to First 4 Day's Data

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 camera and purchasing items with Amazon Dash. 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 §3.1.1. 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_DashButton, 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, with results shown in Table 9. 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 appears to have contact `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.

4.2 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 §4.1. We also validate our claim that DNS-based detection can be applied to old network measurements by showing IoT device-to-server-name mapping 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 in terms of 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 false device merge. Recall in §3.1.1, we merge TPLink_SmartPlug 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 it-

Server Used	Detection Correctness	Server Learnt Back /Dropped (Ratio)
100%	100%	—
90%	100%	5/8 (63%)
80%	96%	6/15 (40%)
70%	96%	10/22 (46%)
60%	92%	11/29 (38%)
50%	96%	21/36 (58%)

Table 10: Resilience of detection and server learning

eration of detection, we learn a new server deventry.tplinkcloud.com for TPLink_Plug/Bulb (from a detected TPLink_LightBulb, as shown by groundtruth). However with now 2 server names mapped to TPLink_Plug/Bulb, we see one less detection of it in second iteration (groundtruth shows a TPLink_SmartPlug becomes un-detected). This reduced detection suggests TPLink_LightBulb and TPLink_SmartPlug 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 false device merge and re-discover the missed TPLink_SmartPlug in subsequent detections.

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.

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

Table 10 shows detection accuracy are fairly stable: with 50 % servers dropped we still have 96% accuracy. We believe two reasons cause this high accuracy: 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. The server learned back ratio are also 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_SmartPlug 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_SmartPlug (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-to-server-name mapping are stable over time. We show 8 of our 10 IoT devices (Table 1) and the 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).

5. RELATED WORK

Prior groups considered detection of IoT devices:

Traffic analysis: IoTScanner detects LAN-side devices by passive measurement within the LAN [23]. They intercept wireless signals (WiFi, Bluetooth and Zigbee packets) and identify IoT devices by packets' MAC addresses. They depend on MAC addresses to identify devices, so their work requires LAN access and cannot generalize to detection from Internet-wide traffic. In comparison, our three methods apply to whatever parts of the Internet that are visible in available network measurements, and are able to categorize devices based on what device servers they visit or TLS certificate they use.

Work from the University of New South Wales characterizes traffic of 21 IoT devices using traffic metadata, including sleeping time, average packet

size, and number of DNS requests sent [24]. They briefly discuss identifying LAN-side IoT devices from LAN-side measurement using traffic metadata. Our work uses different detection signals: packet exchanges with particular device servers and TLS certificate for IoT remote access rather than their types of metadata. Our IP-based and DNS-based methods cover IoT devices both with public IP and behind NAT from observation of network traffic. Our certificate-based method covers all HTTPS-Accessible IoT devices on public Internet by crawling TLS certificates in IPv4 space.

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) [22]. 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 [5].

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 even on traffic from 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.

6. 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 HTTP-Accessible IoT devices from their TLS Certificates. We apply our methods to multiple real-world network measurements. Our IP-based algorithm finds at least 35 IoT devices in a college Campus and 122 IoT devices in customers of a regional IXP. Our DNS-based algorithm finds about 7× growth in ISP-level deployment of 26 device types from 2013 to 2017 and confirms similar increasing 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 [24]. We thank Paul Vixie for providing historical DNS data from Farsight [21]. We especially thank Mark Allman for sharing his CCZ DNS Transactions datasets [2] and help run our code on partially unencrypted 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.

7. REFERENCES

[1] Acar, G., Apthorpe, N., Feamster, N., Huang, D. Y., Frank, and Narayanan, A. IoT Inspector Project from Princeton University. https://iot-inspector.princeton.edu/.

[2] Allman, M. Case Connection Zone DNS Transactions, January 2018 (latest release). http://www.icir.org/mallman/data.html.

[3] DAHUA. Important Message from Foscam Digital Technologies Regarding US Sales and Service. http://foscam.us/products.html/.

[4] DNS-OARC. DNS-OARC DITL Datasets. https://www.dns-oarc.net/oarc/data/ditl.

[5] Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., and Halderman, J. A. A search engine backed by Internet-wide scanning. In *Proceedings of the ACM Conference on Computer and Communications Security* (Denver, CO, USA, Oct. 2015), ACM, pp. 542–553.

[6] Dyn. Dyn analysis summary of Friday October 21 attack. http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[7] Gartner. The Internet of Things units installed base from 2014 to 2020. https://www.statista.com/statistics/370350/internet-of-things-installed-base-by-category/.

[8] GlobalInfoResearch. Global IP camera market by manufacturers, countries, type and application, forecast to 2022. https://www.wiseguyreports.com/reports/1273832-global-ip-camera-market-by-manufacturers-countries-type-and-application-forecast.

[9] GlobalInfoResearch. Network video recorder NVR industry to 2022 market, capacity, generation, investment, trends, regulations and opportunities. http://www.kusi.com/story/35329831/network-video-recorder-nvr-industry-to-2022-market-capacity-generation-investment-trends-regulations-and-opportunities.

[10] Guo, H., and Heidemann, J. IoT traces from 10 device we purchased. https://ant.isi.edu/datasets/iot/.

[11] Guo, H., and Heidemann, J. IP-based IoT device detection. In *Proceedings of the 2nd Workshop on IoT Security and Privacy* (aug 2018).

[12] Krebs, B. KrebsOnSecurity hit with record DDoS. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/.

[13] Kurkowski, J. Python domain extraction library tldextract. https://pypi.python.org/pypi/tldextract.

[14] Loshin, P. Details emerging on Dyn DNS DDoS attack, Mirai IoT botnet. blog http://searchsecurity.techtarget.com/news/450401962/Details-emerging-on-Dyn-DNS-DDoS-attack-Mirai-IoT-botnet, Oct. 2016.

[15] Morales, C. NETSCOUT Arbor confirms 1.7 Tbps DDoS attack; the terabit attack era is upon us. Arbor blog https://asert.arbornetworks.com/netscout-arbor-confirms-1-7-tbps-ddos-attack-terabit-attack-era-upon-us/, Mar. 2018.

[16] Motherboard. How 1.5 million connected cameras were hijacked to make an unprecedented botnet. https://motherboard.vice.com/en_us/article/8q8dab/15-million-connected-cameras-ddos-botnet-brian-krebs.

[17] Mozilla. Public suffix list from Mozilla foundation. https://www.publicsuffix.org/.

[18] No-IP. Domain names provided by No-IP. http://www.noip.com/support/faq/free-dynamic-dns-domains/.

[19] OVH. OVH news - the DDoS that didn't break the camel's VAC. https://www.ovh.com/us/news/articles/a2367.the-ddos-that-didnt-break-the-camels-vac.

[20] SCIP. Belkin Wemo switch communications analysis. https://www.scip.ch/en/?labs.20160218.

[21] Security, F. Passive DNS historical Internet database: Farsight DNSDB. https://www.farsightsecurity.com/solutions/dnsdb/.

[22] Shodan. Shodan search engine front page. https://www.shodan.io/.

[23] Siby, S., Maiti, R. R., and Tippenhauer, N. O. IoTscanner: Detecting privacy threats in IoT neighborhoods. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security* (New York, NY, USA, 2017), IoTPTS '17, ACM, pp. 23–30.

[24] Sivanathan, A., Sherratt, D., Gharakheili, H. H., Radford, A., Wijenayake, C., Vishwanath, A., and Sivaraman, V. Characterizing and classifying IoT traffic in smart cities and campuses. In *Proceedings of the IEEE Infocom Workshop on Smart Cities and Urban Computing* (May 2017), pp. 559–564.

[25] USC/LANDER. Day In the Life of The Internet (DITL) 2013-05-28, 2014-04-28, 2015-04-13, 2016-04-05, 2016-10-04, 2017-04-11, 2017-09-19 and 2018-04-10 datasets. provided by USC/B-Root operations with USC/LANDER project (http://www.isi.edu/ant/lander).

[26] 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).

[27] ZMap. ZMap 443 HTTPS SSL full IPv4 datasets. https://censys.io/data/443-https-ssl_3-full_ipv4.

A. IP CAMERA AND NVR DETECTION BY COUNTRY

In §3.3 we geo-locate our IP camera and NVR detection results, finding them in 199 different countries. Here we examine what devices are in each country to gain confidence in what we detect.

Table 11 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 7.)

We find manufacturers prefer different operating

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 11: Detected IP cameras and NVRs by Countries

regions. We believe these preferences are related to their business strategies. While Dahua, Foscam and Hikvision are global,the latter two show substantially more deployment in the U.S. and China, respectively. Amcrest (formerly Foscam U.S. [3]) is almost exclusive to the American market. The German company Mobotix, while is present in Europe and America, seems completely absent from Asian markets.

This alignment of detection of deployments with company's locality is one use of our detection method.