IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis

Roberto Perdisci*[†‡], Thomas Papastergiou*[‡], Omar Alrawi[‡] and Manos Antonakakis[‡]
[‡]*Georgia Institute of Technology,* [†]*University of Georgia*
{perdisci,tpapastergiou,alrawi,manos}@gatech.edu

***Abstract*—Being able to enumerate potentially vulnerable IoT devices across the Internet is important, because it allows for assessing global Internet risks and enables network operators to check the hygiene of their own networks. To this end, in this paper we propose `IoTFinder`, a system for efficient, large-scale *passive* identification of IoT devices. Specifically, we leverage distributed passive DNS data collection, and develop a machine learning-based system that aims to accurately identify a large variety of IoT devices based solely on their *DNS fingerprints*. Our system is independent of whether the devices reside behind a NAT or other middleboxes, or whether they are assigned an IPv4 or IPv6 address.**

We design `IoTFinder` as a multi-label classifier, and evaluate its accuracy in several different settings, including computing detection results over a third-party IoT traffic dataset and DNS traffic collected at a US-based ISP hosting more than 40 million clients. The experimental results show that our approach allows for accurately detecting many diverse IoT devices, even when they are hosted behind a NAT and their traffic is "mixed" with traffic generated by other IoT and non-IoT devices hosted in the same local network.

***Index Terms*—IoT Security, Traffic Modeling, Passive DNS**

1. Introduction

The number of IoT devices connected to the Internet, such as cameras, voice-activated assistants, network-attached storage devices, smart appliances, etc., has been growing at an increasingly accelerating pace [1], [2]. Unfortunately, partly due to market forces that push down the cost of such devices, even well-known elementary security mechanism are often neglected by IoT device developers and vendors. This has caused a new security crisis of sorts, which has been highlighted by recent major Internet-wide security incidents originating from massive numbers of compromised IoT devices [3]–[5].

It is therefore important to enumerate potentially vulnerable IoT devices across the Internet, as a way to estimate global Internet risks and provide a way for network operators to assess the hygiene of their own networks. For instance, systems such as Censys [6], [7] and Shodan [8] perform periodic scans of the entire IPv4 space, using active probing techniques to identify network services reachable from the public Internet. These services leverage banner grabbing and other fingerprinting approaches to identify and enumerate the type of devices that expose those network services, which includes identifying exposed IoT devices. This allows organizations to assess what others can learn about their networks, thus gaining a better understanding of the attack surface they expose.

Unfortunately, active probing is unable to identify IoT devices that are hosted behind some middleboxes, such as NATs or firewalls. At the same time, mapping these "hidden" IoT devices is important. For example, large Internet service providers (ISPs) may want to build a map of where specific IoT devices are located within their networks and help the hosting clients secure those devices, especially after new IoT vulnerabilities are discovered.

Besides the barriers imposed by some midleboxes, probing-based IoT device discovery is also hindered by the advent of IPv6, which has made exhaustively scanning the entire IP space impractical. While some approaches have been suggested to mitigate this issue, enumerating all possible active IPv6 devices on the Internet, including discovering IPv6 IoT devices, remains a challenge [9], [10]. At the same time, the role of NATs may fade away in IPv6, and there is evidence that devices taking public IPv6 addresses by default may be unintentionally exposed to the Internet with no filtering [9], which may in turn suddenly make previously unreachable IoT devices remotely exploitable.

To address the challenges outlined above, in this paper we propose `IoTFinder`, a system for efficient, large-scale *passive* identification of IoT devices. Specifically, we leverage distributed passive DNS data collection, and develop a machine learning-based system that aims to accurately identify a large variety of IoT devices, with granularity up to device models. Furthermore, `IoTFinder` is independent of whether the devices reside behind a NAT (or other middleboxes, such as firewalls) or are assigned an IPv4 or IPv6 address. It is also worth noting that, while not readily available to everyone, large-scale DNS data collected from globally distributed resolvers has been used in research as well as in many security applications in both the industry and academia [11]–[17]. There also exist companies that focus on collecting, aggregating and offering DNS data as one of their main products (e.g, [18]).

We design `IoTFinder` to meet the following desirable properties:

- *Fully-automated fingerprint learning*: We aim to automatically learn accurate DNS-based statistical fingerprints for IoT device identification, including automatically learning an optimal detection threshold that aims to maximize true positives while limiting false positives below a configurable tolerable rate.
- *One fingerprint per IoT device*: We aim to learn individual *binary classification* fingerprints, one per each IoT device. Namely, given a DNS traffic trace D, a fingerprint f_k learned from device I_k can be viewed as a binary classifier that reveals

* *The first two authors contributed equally to this work.*

DOI 10.1109/EuroSP48549.2020.00037

whether device I_k generated (or contributed to) the DNS activities seen in D. Unlike a multi-class classification approach, these individual fingerprints allow us to easily perform multi-label classification, enabling the identification of multiple different IoT devices that may be co-hosted behind the same (NATed) IP address.

- *Accurately identifying IoT devices in mixed network traffic*: Often, multiple IoT devices are co-hosted behind the same WiFi access point/NAT along with many other general-purpose (non-IoT) devices, including Windows PCs, Mac OS and Linux machines, tablets, smartphones, etc. Our DNS-based statistical fingerprints aim to accurately identify what IoT devices are hosted behind a given client IP address, regardless of whether their traffic is seen from outside their local network as being mixed-in with other IoT or non-IoT traffic.
- *Explainable results*: We aim to avoid complex models whose results are difficult to interpret. Instead, we favor statistical IoT device fingerprints that are easily interpretable. This allows us not only to understand the learned IoT models themselves, but also to analyze the fingerprint matching results and investigate the causes of possible false positives and negatives.
- *Efficient matching*: Because our objective is to enable global-scale identification of IoT devices, we aim to develop our fingerprint matching algorithm to be highly efficient. For instance, we aim to match tens of different IoT device fingerprints against DNS traffic generated by tens of millions of clients in a matter of minutes.

Our system is generic, in that it is able to *automatically learn* detailed DNS fingerprints for IoT devices based on samples of real-world device-generated traffic without relying on manually crafted rules or constraints. Specifically, we automatically model the behavior of more than 50 IoT devices from a variety of manufacturers. Furthermore, we develop a method for collecting negative ground truth (i.e., labeled DNS traffic from non-IoT devices) at scale, allowing us to quantify and characterize possible false positives over large numbers of real-world non-IoT devices, including Windows PCs, Mac OS laptops, iPhones, iPads, Android mobile devices, etc.

After systematically quantifying `IoTFinder`'s true and false positives and providing a method for automatically tuning the detection threshold of each DNS fingerprint, we match our IoT behavior models over very large DNS traffic datasets collected from a major ISP network with tens of millions of clients, allowing us to estimate the population of these IoT devices across large sections of the Internet.

In summary, we make the following contributions:

- We propose a novel system called `IoTFinder` for efficient large-scale detection of in-the-wild IoT devices. `IoTFinder` is designed to automatically learn statistical DNS traffic fingerprints from real IoT devices, and to efficiently match the derived fingerprints over very large volumes of DNS traffic collected from many geographically diverse network locations.
- Overall, `IoTFinder` works as a multi-label classifier that is able to detect IoT devices even when they are hosted behind a NAT and their traffic is "mixed" with traffic generated by other IoT and non-IoT devices hosted in the same local network.
- To evaluate the accuracy of our statistical models, we perform a detailed evaluation of our IoT fingerprints in several different settings, including computing detection results over a third-party IoT traffic dataset. These detailed experiments allow us to highlight the advantages of the proposed approach as well as to point out possible limitations.
- We also apply our IoT fingerprints over DNS traffic collected at a US-based ISP hosting more than 40 million clients, to assess our system's performance. Our results show that we can efficiently process one entire day of ISP-level DNS traffic and test all our IoT fingerprints in a limited amount of time (slightly more than one hour).

2. Intuitions and Approach Overview

In this section, we provide the intuitions behind our approach for large-scale passive detection of IoT devices based on DNS traffic analysis. We first briefly describe how we gather ground truth data to learn our IoT detection models (Section 4.1), so that we can provide context for the motivating examples discussed in Section 2.2. Then, we provide an overview of how our system learns to recognize the DNS-behavior of IoT devices (Section 2.3), by drawing an analogy to document retrieval systems.

2.1. IoT Data Gathering

To enable the training and testing of our model, we have established a large IoT lab consisting of multiple voice assistants, cameras, smart speaker, IoT hubs, lights, TVs, game consoles, smart appliences, etc. We continuously record all traffic sent and received by each device, as depicted in Figure 1. The devices are deployed in a lab with human presence, in which cameras record movement, voice-based assistants listen to discussions and are from time to time used/activated by the people in the lab, thermostats sense changes in the room temperature, appliances (e.g., a Roomba vacuum cleaner) are occasionally used (e.g, to clean the lab floor), etc. In this paper, we focus specifically on the DNS traffic generated by the IoT devices. The IoT DNS dataset we use in our experiments spans approximately 1.5 months, and includes 53 active IoT devices from many different vendors (see Section 4.1 for more details).

2.2. Motivating Examples

Figure 2 shows an example of how different devices tend to query different (potentially overlapping) sets of domain names with different frequency, in a 24 hours period, based on data we collected as described in Section 4.1.1 (the figure only shows the DNS behavior of six devices, due to space constraints).

Based on the DNS activity generated by IoT devices, one may be tempted to build the following simplistic detection model (a somewhat similar model was proposed

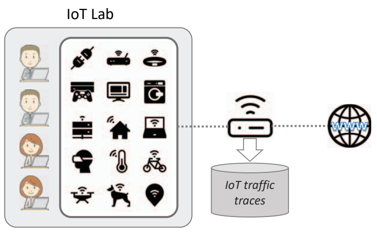


Figure 1: IoT traffic collection infrastructure.

in [19]): given the set of domains queried by a specific IoT device and a passive DNS trace collected from a third-party network, if a DNS client (e.g., identified by an anonymous ID or IP address) in the trace queries any of the domains in the IoT device domains set, label that client's IP (or anonymous ID) as hosting the IoT device behind it. Unfortunately, this naive model is prone to numerous false positives. For instance, device #5 (Philips HUE hub) in Figure 2 queries, among others, www2.meethue.com, which is a publicly accessible website containing product information. Other devices (not shown due to space constraints) query popular domains such as www.google.com, pool.ntp.org, etc. Previous work [19] has proposed to deal with such cases by probing the domains to determine whether they are "human facing" (i.e., whether they contain human-accessible content), thus filtering out domains such as www2.meethue.com to avoid false positives. However, the frequency with which www2.meethue.com is queried by device #5 is hardly characteristic of a human visiting the meethue.com website. Therefore, discarding such information would neglect the potentially discriminative classification power that comes with it. Similarly, the approach proposed in [19] would discard fireoscaptiveportal.com from consideration in the detection of device #57 (Amazon Echo Dot), because the domain does not contain a string representing either the vendor name or the device model name, even though that domain is queried with high frequency. For the same reasons, [19] would discard both domains queried by device #33 (Belkin WeMo Crockpot), thus preventing its correct identification. Furthermore, using single domain names for detection can cause significant confusion among devices that share some DNS behavior (e.g., different device models from the same vendor), which is another limitation affecting the system proposed in [19].

At a high level, Figure 2 provides the intuition that, unlike single domains, the *combination* of domains and related *frequency* of DNS queries issued by different IoT devices are often quite specific. This observation may indeed allow us to build a set of discriminative behavioral fingerprints that can identify different IoT device models and are unlikely to trigger a significant number of false positives, when matched against a generic client's DNS behavior. We therefore leverage the above intuition to build a system that can *automatically* learn accurate DNS-based IoT fingerprints, without the need of making strong assumptions on the format of the domains a device may query or of manaully deriving filtering heuristics to reduce false positives (see Section 3 for details).

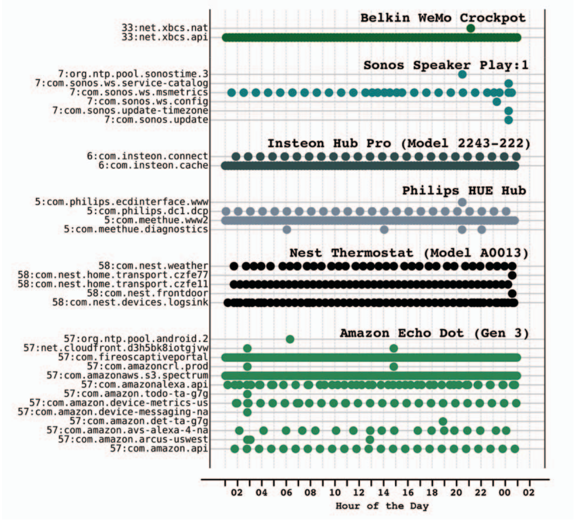


Figure 2: Timeline of IoT domain queries over 24 hours. The number on the left of each domain name string is a unique IoT lab device identifier (5: Philips HUE hub; 6: Insteon hub; 7: Sono speaker; 33: Belkin WeMo Crockpot; 58: Nest thermostat; 57: Amazon Echo Dot (3rd Gen)). Each horizontal line represents a domain name queried by an IoT lab device, and each dot represents the time of a DNS query. Vertical dashed lines mark a one hour window.

2.3. Approach Overview

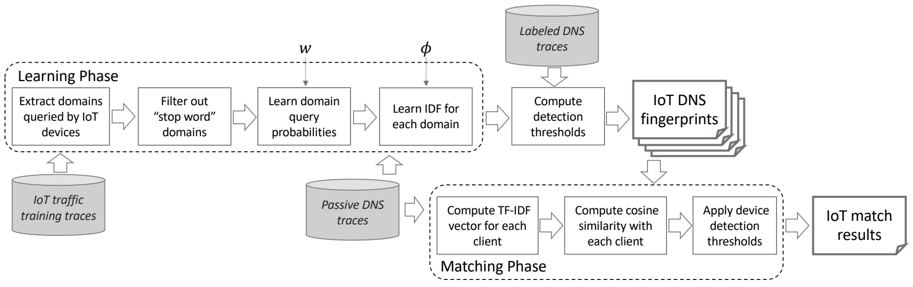
To build efficient DNS-based IoT device fingerprints, we first draw an analogy between IoT device detection and *document retrieval* [20]. Let C_i be a generic device (IoT or not), represented in practice by its IP (v4 or v6) address. Also, let d_{ij} be a domain name queried by device C_i, and f_{ij} be the occurrence frequency of d_{ij} in C_i's DNS traffic. Namely, given a time interval Δt, f_{ij} expresses the number of DNS queries to d_{ij} issued by C_i. We can think about $C_i = \{(d_{ij}, f_{ij})\}_j$ as a *document* containing a set of *terms* d_{ij}, each appearing in the document with their own respective term frequency f_{ij}.

Similarly, let Q_k be an IoT device, q_{kj} be a domain name queried by Q_k and f_{kj} be the frequency with which q_{kj} is queried by Q_k. Continuing the analogy with document retrieval, we can think of $Q_k = \{(q_{kj}, f_{kj})\}_j$ as *search query terms*, and the task of identifying IoT devices as retrieving all documents C_i that are *similar* to the *search query* Q_k, with relevance ranked according to their similarity scores.

Following the above analogy, to compute the similarity between the DNS behavior of an IoT device Q_k and that of a generic device C_i we can first translate each Q_k and C_i into their respective TF-IDF (i.e., term frequency - inverse document frequency) feature vector representations [20], and then measure the similarity (e.g., the cosine similarity) between the resulting feature vectors.

An overview of the approach used by our `IoTFinder` system for learning and matching TF-IDF feature vectors for IoT devices is shown in Figure 3, which we will describe in detail in Section 3.

Challenges: There are some important challenges related to the above analogy between our IoT device modeling and document retrieval. First, our design goal of being



Learning Phase
Extract domains queried by IoT devices
Filter out "stop word" domains
Learn domain query probabilities
Learn IDF for each domain
w
ϕ
Labeled DNS traces
Compute detection thresholds
IoT DNS fingerprints
IoT traffic training traces
Passive DNS traces
Compute TF-IDF vector for each client
Compute cosine similarity with each client
Apply device detection thresholds
IoT match results
Matching Phase

Figure 3: Learning and Matching DNS-based IoT fingerprints.

able to match an IoT device whether or not it is hosted behind a NAT along with several other devices does not immediately fit the document retrieval scenario. One may think that this could be analogous to matching a query against a document containing multiple topics. However, in the basic document retrieval case, documents that focus on one single topic that is similar to the query may tend to be favored (ranked higher), compared to documents with multiple topics. On the contrary, we do not wish to penalize matching an IoT device behind a NAT simply because it is hosted along other (IoT and non-IoT) devices behind the same IP address. Furthermore, some routers/NATs act as DNS forwarders with their own DNS cache policies. This may cause a discrepancy between the occurrence frequency of domain names queried by a device C_i and that learned by the reference model for a target IoT device Q_k, even if C_i and Q_k do represent the DNS behavior of (two instances of) the same IoT device. This discrepancy could be further aggravated by possible packet loss at the DNS traffic collection point, and device usage patterns (e.g., some devices, such as smart appliances, may be turned on and off). Sections 3.1 and 3.2 explain how we adapt our IoT device detection fingerprints to address the challenges outlined above while maintaining the high-level analogy with document retrieval.

3. System Details

We now provide a detailed description of `IoTFinder`'s fingerprint learning and matching processes summarized in Figure 3.

3.1. Learning DNS-based IoT fingerprints

Given a set of IoT traffic traces collected over a period of time T_l (e.g., several days) from I different IoT devices, we first extract the set of domain names queried by each device in our IoT lab (see Section 4.1). To model the DNS behavior of an IoT device, we proceed as follows. Let $Q_k = \{q_{kj}\}_{j=1}^{m_k}$ represent an IoT device that, during T_l, queries m_k distinct domain names, q_{kj}. First we divide the observation time T_l into non-overlapping time windows of length w (for simplicity, we will assume T_l is a multiple of w). Given a domain q_{kj}, for each of the $N_w = \frac{T_l}{w}$ time windows, we determine whether q_{kj} was queried or not during that window. Now, assume we observed at least one DNS query to q_{kj} in $n_{q_{kj}}$ out of N_w time windows. We then say that device Q_k queries q_{kj} with probability $p_{kj} = \frac{n_{q_{kj}}}{N_w}$. In other words, we discretize time, and then compute the probability that Q_k will query q_{kj} at least once in any time window of length w. This results into a statistical fingerprint $P_k = \{(q_{kj}, p_{kj})\}_{j=1}^{m_k}$ for each IoT device Q_k, which we will map to *term frequency* estimates at test time. The mapping between query probabilities and term frequencies will depend on the length of the window of traffic considered at test time, as explained in Section 3.2.

The time discretization process explained above allows us to compensate for possible short-term variations in the query frequency among IoT devices of the same vendor and model. For example, assume we have two identical IoT devices, I_a and I_b, and that I_a is deployed behind a caching DNS forwarder (e.g., a NAT that acts as such), whereas I_b is connected to an ISP network either directly or through a non-caching middlebox device. If we observe the DNS traffic from the ISP's DNS resolvers (before their own DNS cache), we may end up measuring different query frequencies from I_a and I_b, due to different DNS caching behavior implemented by the middlebox in front of the two devices. By setting the length of window w to be larger than the time-to-live (TTL) of DNS resource records (e.g., `A` and `AAAA` records) for domains queried by I_a and I_b, we effectively approximate the behavior of a DNS cache that acts equally on both devices. This has a "normalizing" effect that allows for a more accurate IoT fingerprint matching. We discuss how to chose an appropriate value of w in Section 4 (see also Figure 6).

As a second learning step, we compute the *inverse document frequency* (IDF), which essentially estimates the inverse "popularity" of each IoT-queried domain across global DNS traces. More formally, let $\mathcal{Q} = \{q_i\}_{i=1}^{m}$ be the set of all distinct domain names queried by any of the IoT devices represented in our IoT lab dataset; namely, $\mathcal{Q} = \bigcup_k \{q_{kj}\}_{j=1}^{m_k}$. Also, let T_p be a given observation period within which passive DNS traces are collected (e.g., one or more days), $N_c(q_i)$ be the number of clients that query an IoT domain $q_i \in \mathcal{Q}$ during T_p, and N_c be the total number of clients that query at least one IoT domain in $\mathcal{Q}$ during T_p. We define $IDF(q_i) = log(1 + \frac{N_c}{N_c(q_i)+1})$. That is, the less common a domain q_i is among all clients that have a chance to match an IoT device, the higher its inverse document frequency (i.e., the higher its specificity).

In practice, we first filter out from our modeling those domain names that are known to have a very high popularity, such as the top 100 domains according to `Alexa.com`. This is akin to filtering out stop words, such as "the", "a", etc., in the document retrieval analogy. More precisely, given a top domain d to be filtered (e.g., `google.com`), we filter out d and `www.`d. This filtering serves two purposes: (i) decreasing the number of clients C_i to be considered for matching at test time, thus making fingerprint matching more efficient; and (ii) decreasing the probability of triggering false positives, given that such highly popular domains are queried by most non-IoT devices and thus may contribute to spurious matches.

In addition to learning a model of the DNS traffic generated by each IoT device, as explained above, we also automatically learn the detection threshold that will allow us to identify specific IoT devices over never-before-seen passive DNS traces with a maximum tolerable fraction ϕ of false positives. We will explain the threshold learning process in Section 3.3, after explaining how fingerprint matching works.

At the end of the learning process, we obtain a statistical fingerprint for each IoT device. An IoT DNS fingerprint will store the following information: (a) the IoT domain query probabilities, $P_k = \{(q_{kj}, p_{kj})\}_{j=1}^{m_k}$, for IoT device Q_k; (b) the value of the time window w used to compute P_k; (c) the IDF value for each domain, $\{IDF(q_{kj})\}_{j=1}^{m_k}$; and (d) the detection threshold, θ_k, computed based on the maximum tolerable false positive rate per device, ϕ.

3.2. Matching DNS-based IoT fingerprints

We now explain how the IoT device fingerprints can be matched against future passive DNS traces. Let $P_k = \{(q_{kj}, p_{kj})\}_{j=1}^{m_k}$ be the statistical fingerprint for IoT device Q_k, derived as explained in Section 3.1. Also, let T_t be a time window in which DNS traffic is passively observed (e.g., one day), and C_i be a client whose DNS queries are included in the DNS traffic. Our goal is to compute a similarity score, s_{ki}, between fingerprint P_k and the DNS behavior of C_i.

To leverage a TF-IDF vector representation, as explained in Section 2.3, we need to compute a *term frequency* vector for each IoT device. To do so, we map the query probabilities in P_k to the observation period T_t. To this end, we first compute the number of time windows $N_t = \lceil \frac{T_t}{w} \rceil$, where w is the same time window parameter used in learning P_k. Then, we set $f_{kj} = p_{kj} * N_t$, and the TF vector $V_k = \{(q_{kj}, f_{kj})\}_{j=1}^{m_k}$. Namely, f_{kj} is the expected number of time windows in which IoT device k would query domain q_{kj} during T_t. Finally, we multiply f_{kj} by the previously learned IDF (see Section 3.1) of the respective domain q_{kj}, and obtain the corresponding TF-IDF feature value $\psi_{kj} = f_{kj} \cdot IDF(q_{kj})$. We then use $\Psi_k = \{(q_{kj}, \psi_{kj})\}_{j=1}^{m_k}$ to represent the TF-IDF vector for IoT device Q_k over time T_t.

We now need to compute the TF-IDF vector for a client C_i. Let us assume that C_i queries a set of n_i distinct domain names d_{ij} with occurrence frequency f_{ij} during the observation period T_t. Namely, $C_i = \{(d_{ij}, f_{ij})\}_{j=1}^{n_i}$, where f_{ij} is computed as before by counting the number of time windows of length w in which a query to d_{ij} was observed. To compute the similarity between an IoT TF-IDF vector Ψ_k and the behavior of a client C_i, we then proceed as follows. First, we "project" C_i into the space of m_k domains in Ψ_k. In practice, we compute a new vector $C'_i = \{(q_{kl}, f'_{kl})\}_{l=1}^{m_k}$, where the set of domains $\{q_{kl}\}_{l=1}^{m_k}$ is the same set of m_k domains represented in the IoT device vector Ψ_k. We then set $f'_{kl} = f_{ij}$ if $q_{kl} = d_{ij}$. If no domain name d_{ij} matches q_{kl}, we set $f'_{kl} = 0$. In other words, we take all domains queried by client C_i that were also queried by IoT device Q_k, and set their TF values to reflect the frequency with which C_i queried them during time T_t. If a domain queried by IoT device Q_k was not observed in C_i's behavior, we set its frequency to zero. Finally, we compute the TF-IDF vector $\Gamma_i = \{(q_{kl}, \gamma_{kl})\}_{l=1}^{m_k}$, where $\gamma_{kl} = f'_{kl} \cdot IDF(q_{kl})$.

This "projection" of C_i's behavior onto Q_k's domain names space allows us to achieve one of the main goals of our work. Namely, it allows us to avoid penalizing the similarity score between Q_k and C_i simply because C_i represents the DNS traffic of multiple devices (possibly including several other IoT and non-IoT devices) that coexist behind the same IP address (e.g., behind a NAT).

Consequently, we now have that Ψ_k and Γ_i span the same vector space. Specifically, let $\psi = [\psi_{kj}]$ and $\gamma = [\gamma_{kj}]$ be two vectors containing the TF-IDF features of Ψ_k and Γ_i, respectively. We compute the *matching score* as $s(\Psi_k, \Gamma_i) = \frac{\psi \cdot \gamma}{||\psi|| \cdot ||\gamma||}$ (i.e., we compute their cosine similarity). Then, if $s(\Psi_k, \Gamma_i) \geqslant \theta_k$, we say that the DNS behavior of client C_i matches that of IoT device Q_k. Here, θ_k is a threshold that is automatically computed to limit the number of false positives to a predefined maximum ϕ.

3.3. Learning Device Detection Thresholds

To learn each device-specific detection threshold, θ_k, we proceed as follows. First, we collect a large dataset $\mathcal{D}$ of previously unseen (i.e., not seen during training) *labeled* passive DNS traces, as explained in Section 4.1.3. In practice, $\mathcal{D}$ provides us with traffic traces from a large set (e.g., tens of thousands) of non-IoT devices, which form our *negative ground truth*. Then, we take the DNS behavior of each client C_i in $\mathcal{D}$, and match it against each of our previously learned IoT device fingerprints. As explained in Section 3.2, this allows us to compute the cosine similarly, $s(\Psi_k, \Gamma_i)$, between each fingerprint and every non-IoT device in $\mathcal{D}$. Using this, we can also compute how the false positives vary by tuning the detection threshold θ_k. Similarly, we consider previously unseen DNS traffic generated by IoT device Q_k, which we then can use to compute the true positive rate of each IoT device fingerprint Ψ_k, as a function of the detection threshold θ_k.

Given a maximum tolerable false positive rate ϕ, our goal is to find a value for each detection threshold θ_k so to meet the two following requirements: (1) fingerprint Ψ_k generates a false positive rate $F_k \leqslant \phi$, when matched against future DNS traces; and (2) maximize the true positive rate while keeping $F_k \leqslant \phi$. To meet both constraints at the same time, we devised the heuristic approach described below.

First, we compute the ROC curve (i.e., the trade-off between true and false positives as the detection threshold varies) by matching fingerprint Ψ_k over datasets $\mathcal{D}$ and device Q_k. Figure 4 shows a portion of an example ROC curve, to make our explanation easier to follow. Notice that in reality the ROC curve is not continuous. Instead, the curve is built in practice as a linear interpolation between all possible operating points, namely the values of true and false positives given by each possible detection threshold.

Given the ROC curve, we first find the operating point on the curve that corresponds to a false positive rate $f = \phi$. Notice that, as shown in Figure 4, such a point

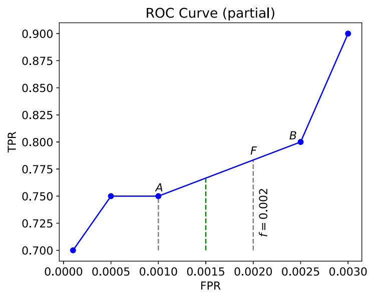


Figure 4: Example of (partial) ROC curve with linear interpolation

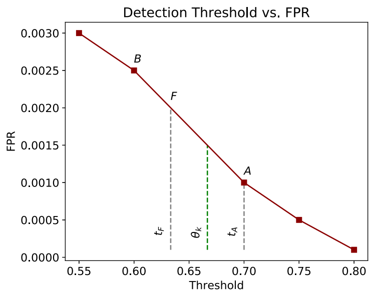


Figure 5: Linear interpolation of FPR changes as a function of the threshold

may not actually exist, but can be estimated via linear interpolation. Once f is computed, we can derive the detection threshold, t_F, related to the corresponding point on the ROC curve. The example in Figure 5 is derived from the (partial) ROC curve in Figure 4, and shows the relationship between the detection threshold and the false positive rate. Again, we can estimate the threshold t_F corresponding to f via interpolation.

We may then expect that setting $\theta_k = t_F$ would conclude our search for a detection threshold that meets the two requirements stated earlier. However, we should notice that there is uncertainty regarding how fast the false positives may actually rise between points A and B. In other words, in practice the false positive rate may grow faster than linearly between A and B, when fingerprint Ψ_k is tested on future data. Therefore, we prefer to conservatively set the threshold somewhat above t_F, to give ourselves some margin. To this end, we proceed as follows: we first consider the empirically computed points A and F as in Figure 4, and map them to the respective threshold t_A and t_F, as in Figure 5. We then take the middle point, $\theta_k = (t_A + t_F)/2$, and consider a device as detected if the similarity $s(\Psi_k, \Gamma_i) > \theta_k$. Notice that a low value of θ_k indicates that the device is somewhat "easier" to detect, in that a low similarity scores threshold would be sufficient to generate a low false positive rate. On the other hand, a high value of θ_k (e.g., close to $\theta_k = 1$) would indicate that the the device fingerprint for device k must match almost perfectly, for it to have a low false positive rate.

In practice, we also add the constraint that $\theta_k \geqslant \overline{\theta_k}$, and heuristically set $\overline{\theta_k} = 0.5$ to make sure we never match fingerprints with a similarity score that is too low.

3.4. IoT Fingerprints Deployment

Now that we have described in detail how we can learn and match a fingerprint for each single IoT device, we further discuss how these statistical fingerprints can be used to enumerate IoT devices in the wild. First, it is worth remembering that we are interested in passively detecting IoT devices behind a NAT, even when they are co-located with a large number of other IoT devices and general purpose, non-IoT devices. To this end, let i be an in-the-wild DNS client's IP address (e.g., the NAT's public IP address), and $\mathcal{D}_i$ be the DNS traffic generated by i. To determine what IoT devices are hosted behind i, we test each IoT fingerprint we previously learned over $\mathcal{D}_i$, and report the confidence for fingerprint match. Formally, for each device k we check whether $s_{ki} = s(\Psi_k, \Gamma_i) > \theta_k$, and if so report that device k matched client i with confidence s_{ki}. In other words, we use a *multi-label classification* approach, and report all devices that match with a similarity above their respective detection thresholds.

As our main goal is to enable efficient large-scale IoT device identification, even just considering one day of DNS traffic, the number N of distinct client IPs in $\mathcal{D}_i$ can be quite large (in the tens of millions, for our experiments), and yield a huge overall volume of DNS queries (e.g., tens of billions of queries per day). Also, considering the large number M of distinct IoT devices in our lab, computing all M TF-IDF vectors and performing all $N \times M$ fingerprint matches efficiently on a daily basis is highly challenging. To solve this problem, we implemented our feature vectors computation and fingerprint matching in a large Apache Spark cluster, bringing the $N \times M$ fingerprint match compute time down to less than one hour for each day of real-world DNS traffic from a large ISP (see details in Section 4).

4. Evaluation

In this section, we evaluate `IoTFinder`'s performance.

4.1. Data Collection

Our experiments are performed using the datasets described later in this section. For brevity, we will use acronyms to refer to the different datasets, as defined below. Datasets `IoTDNS`, `PDNS`, and `LDNS` contain data collected daily, from the respective sources, between August 2 and September 22 2019, whereas the `TPIoTDNS` third-party dataset contains data collected between October 4 and October 13, 2019.

4.1.1. IoT Lab Traces Dataset (IoTDNS)

`IoTFinder` aims to automatically learn a statistical behavior fingerprint for each IoT device. To enable learning, we collect *ground truth* DNS traffic traces produced by real-world IoT devices in a realistic deployment environment. To this end, we have built an IoT lab consisting of a large number of devices, which include multiple smart plugs, cameras, voice-based home assistants, smart TVs, streaming boxes, IoT hubs, thermostats, gaming stations, smart house appliances, a smart irrigation system, etc. In some cases, such as for deploying the Nest thermostat, we went to the extent of simulating the presence of an AC/Furnace system by connecting the thermostat to a set of 24V AC relays, to simulate the presence of a real cooling/heating system without actually connecting the thermostat to a real load (the thermostat could not otherwise be correctly setup).

IoTDNS contains DNS traffic generated by 53 different active IoT devices, which are listed on the x axis of Figure 7. Notice that some devices have been incrementally added to the lab, compared to the beginning of our data collection period (at the beginning of the data collection period we had 10 fewer active devices). Also, due to operational issues (e.g., temporarily unplugging and moving devices, or other lab reconfiguration issues), some devices may not be active on some specific days. Days of inactivity are removed from our experiments only for the "missing" devices. In other words, all our evaluations for a given IoT device model are relative to days in which that IoT device was actually active. To determine activity, we use conservative heuristics. Specifically, if a device did not generate any DNS, TCP SYN, or more than three non-DNS UDP packets, we consider the device to be inactive (the low threshold for non-DNS UDP packets is to remove spurious packets related to DHCP that may be present even if the device is dormant or malfunctioning).

The devices were deployed in a lab with human presence, in which cameras record movement, voice-based assistants listen to discussions and are from time to time used/activated by the people in the lab, thermostats sense changes in the room temperature, appliances (e.g., a Roomba vacuum cleaner) are occasionally used (e.g, to clean the lab floor), etc. In this environment, all Internet-bound network traffic generated by the IoT devices is continuously captured, as shown in Figure 1. We use the data captured in the lab (along with other data sources, as explained later) to train and test our statistical IoT fingerprints. We plan to share the network traces collected from our IoT lab with the research community.

4.1.2. Passive DNS Traces (PDNS)

We have obtained access to anonymized passive DNS traces from a large Internet service provider (ISP) based in the US. DNS traces are passively collected from a distributed network of local DNS resolvers operated by the ISP, enabling visibility into domain name queries issued daily by more than 40 million Internet-connected devices (including IoT devices) scattered across geographically diverse US locations. We use this passive DNS data for two main purposes: computing the frequency with which domain names queried by IoT devices are queried by generic devices in the wild (see IDF computation in Section 3.1), and estimating the population of each IoT device for which we learn a statistical fingerprint (Section 4.8).

4.1.3. Labeled DNS Traces (LDNS)

We have also gained access to the network traffic of a large university campus based in the US, which we use to collect *labeled* passive DNS traces. Specifically, we use deep packet inspection (DPI) techniques to determine the type of device hosted behind each network client IP, and associate the DNS traffic of these clients with their respective device type labels. For instance, given the traffic originating from a client C_i, we can leverage HTTP traffic generated by C_i to infer the type of hardware and operating system running on it, by parsing User-Agent strings in HTTP requests from C_i. Then, we can label all DNS traffic generated by C_i as belonging to a specific device type, such as a Windows PC, iPhone, iPad, Android phone, Mac OS laptop, etc. We are interested in labeling DNS traffic from non-IoT devices, because we aim to use these labeled DNS traces for two main purposes: (i) to estimate possible false positives generated by our statistical IoT fingerprints (Section 4), and (ii) to automatically compute a detection threshold to be used for global IoT device enumeration (Section 3.3). The details of our DNS traffic labeling approach are described in Section 4.1.5.

LDNS includes data from a daily average of more than 54,000 generic, non-IoT devices, with the average count distribution shown in Table 1. Section 4.1.5 explains the algorithm and heuristics we used to label DNS queries from clients within our campus network.

TABLE 1: Generic (non-IoT) device population – average daily counts

Windows	Linux	Mac	iPad	Android	iPhone
22135	7465	7520	826	6265	9938

4.1.4. Third-Party IoT Device Traffic (TPIoTDNS)

Finally, we obtained network traces collected by a third party entity unrelated to our institution. Specifically, the third party independently setup a small network environment with 6 different active IoT devices from 6 different manufacturers, and a number of non-IoT devices (the traces also included traffic from malware-infected machines, as this was part of a hacking exercise organized by a funding agency). We use these traces as a way to confirm that our fingerprints indeed allow us to detect different IoT devices in different networks, even when multiple devices are hosted behind the same NAT along with other non-IoT devices (e.g., Windows machines, Linux machines, Raspberry PI's, etc.).

4.1.5. Labeling DNS Traces Using DPI

In this Section, we explain how we label DNS traffic for the LDNS dataset. As mentioned in Section 4.1.3, we aim to collect DNS traffic from non-IoT devices because we aim to use these labeled DNS traces for two main purposes: (i) to estimate possible false positives generated by our statistical IoT fingerprints (Section 4), and (ii) to automatically compute a detection threshold (Section 3.3) that will be later used for in-the-wild IoT device enumeration (Section 4.8). To this end, we collect DNS traffic from our own institutional network, and our goal is to label the DNS traffic from each client in the network with the type of device that it represents, such as Windows, Mac OS, or Linux machines, and mobile devices such as Android-based phones or tablets, iPhones and iPads.

To achieve the above goal, we need to address two challenges: (1) correctly associating DNS queries observed by our traffic collection infrastructure with the client that actually generated those queries, and (2) associating a device label to each client, as explained below.

The first challenge arises from the fact that our institutional network is large and is not "flat." We have a traffic collection point at the edge of the network, where we can observe traffic to the main DNS resolver. However, different departments may internally use a caching DNS forwarder that accepts DNS requests from a department's clients and forwards them to the main resolver. Therefore, our data collection infrastructure may record only the IP address of the DNS forwarder, and not the address of the client that actually issued a given query. We address this challenge as follows.

We make the simplifying (but often met) assumption that after a client issues a DNS request it will subsequently attempt a TCP connection with (one of) the resolved IP address(es). Because our traffic collection point can observe the true IP address of outgoing TCP SYN packets (since the DNS forwarder is not involved, and no other middlebox is allowed by our network's policies), we substitute the TCP connection initiation to a resolved IP with a DNS query to the related domain name from which the IP address was resolved. More specifically, let c be a client that sends a TCP SYN packet to a destination IP address p, which the client resolved from domain name d. To map p back to the domain d queried by c, we consider the timestamp, t_s, of the TCP SYN packet sent by c to p, and then walk back over the DNS packets with timestamp $t < t_s$ to find the latest occurrence of a domain name that resolved to p. Notice that this approximates the behavior of a client that directly uses our main DNS resolver, with no other DNS cache present in between the client and the main resolver. Also notice that our DNS query aggregation time window (discussed in Section 3.1 and Section 4.2) has the effect of removing possible noise in the domain query frequencies reconstructed with the method described above.

To address the second challenge (i.e., assigning a device label), we leverage the fact that generic computing devices, such as desktop computer, laptops, and mobile devices, run a variety of applications, some of which generate non-encrypted HTTP traffic that carries a `User-Agent` string. We then use the information in the `User-Agent` for labeling. Due to space constraints, we describe the details of this process in Appendix A.

4.2. Experimental Setup

Unless otherwise stated, the results discussed below are related to IoT device models trained over a period of two consecutive weeks of network traffic from the `IoTDNS` dataset, which are used to compute the domain query probability models P_k as described in Section 3.1. We also set the time window length w to be one hour. This choice is justified by the fact that the vast majority of domain names queried by IoT devices have a TTL below 3600 seconds, as shown in Figure 6. As explained in Section 3.1, discretizing time using w has a sort of normalizing effect, in that it tends to make the timeline for DNS queries comparable whether or not the IoT devices reside behind a NAT and whether or not the NAT implements its own DNS cache.

To compute the *inverse document frequency* (IDF) values for each domain name (see Section 3.1) we use one day of `PDNS` data selected within the same training time period used to compute the P_k models. Furthermore, to evaluate the false positives raised by each IoT detection model, we leverage the labeled DNS traffic in `LDNS`.

4.3. ROC Analysis

As a first experiment, we evaluate the ability of each single IoT device fingerprint to detect future traffic generated

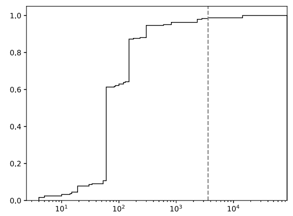


Figure 6: Distribution of TTLs for domain names queried by IoT devices (A records). The dashed vertical lines indicates the 3600 seconds mark.

by the same device, and the number of false positives it may raise over real-world non-IoT traffic. To this end, we compute and analyze the area under the ROC curve per each device classifier, as follows. Let P_k be the statistical model related to IoT device I_k. Also, let D_{train} represent the training data over which we learn P_k, namely two weeks of traffic from I_k (from `IoTDNS`), and D_{test} be the following two weeks of traffic from the same I_k device. We match model P_k over each day of traffic in D_{test}, and thus compute 14 similarity scores (one score per day). These represent the set τ_k of *positive scores*.

Now, let D_{noniot} be one day of traffic extracted from the `LDNS` dataset (picked as the last day within the same two weeks period considered in D_{test}). For each generic, non-IoT device G_i in D_{noniot}, we compute the similarity score between P_k and G_i. These represent the set γ_k of *negative scores*. We then use τ_k and γ_k to compute a *partial* AUC (pAUC) [21], limiting the maximum false positive rate to 10^{-4} (i.e., 0.01%).

We refer to the 4-week period that encompasses training P_k and then matching it over D_{test} and D_{noniot} as Δ_{test}. For each device I_k, we repeat the experiment outlined above for 23 distinct time periods Δ_{test}. Figure 7 shows, per each device, the distribution of the pAUC computed per each device fingerprint. The boxplots show the "spread" of the pAUC, whereas the blue diamonds indicate the median pAUC across the 23 experiments.

To evaluate if our statistical fingerprint tend to perform better on test days that are closer to the end of the training period, we repeat the above 23 experiments while limiting the span of D_{test} to 2 days, 4 days, and 1 week. Due to space limitations, we report the pAUC results related to the 4-day D_{test} scenario in Figure 11 in Appendix. The results show that the median pAUC does not change significantly, compared to the 2-week test periods (Figure 7), thus indicating that our IoT DNS fingerprints do not require very frequent retraining. However, our statistical models could be easily retrained daily, if desired, because training only takes less than two hours on an Apache Spark compute cluster (please see Section 4.8 for more information about the cluster).

4.4. Confusion Among Devices

We now evaluate whether a DNS traffic model P_k trained over traffic from IoT device I_k may also be similar to DNS traffic generated by other IoT devices. To this end, we compute the similarity between P_k and traffic for all other IoT devices (including device I_k itself). In this experiment, we train P_k over a fixed 2-week time window. Then, we consider the following two weeks (i.e., 14 days)

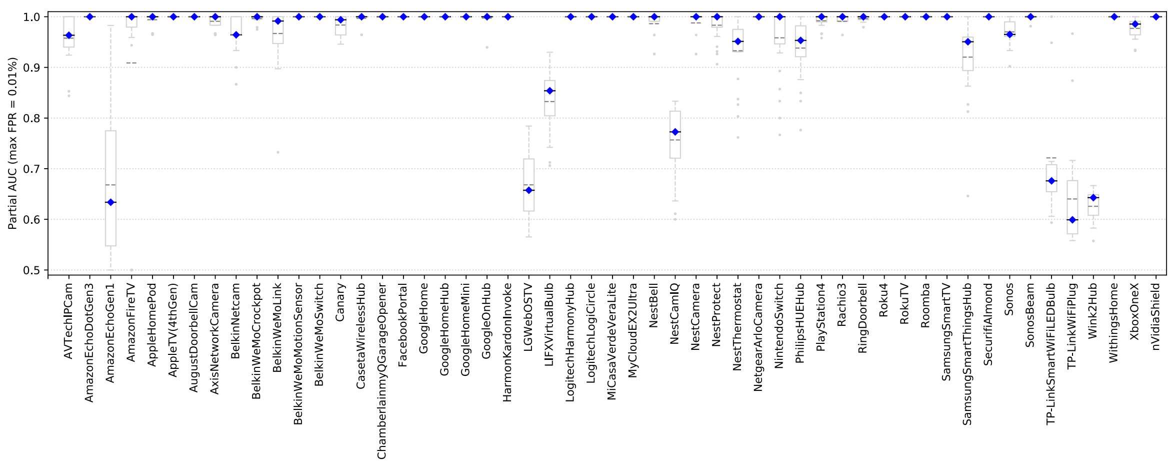


Figure 7: Fingerprints' accuracy: Two weeks training / two week testing. Each fingerprint is trained over two weeks of IoT traffic and tested over traffic collected in the following two weeks. Each test period consists in a different two weeks training / two weeks testing period. Boxplots summarize the pAUC distribution over 20 different training/testing periods. Blue diamonds indicate the median pAUC.

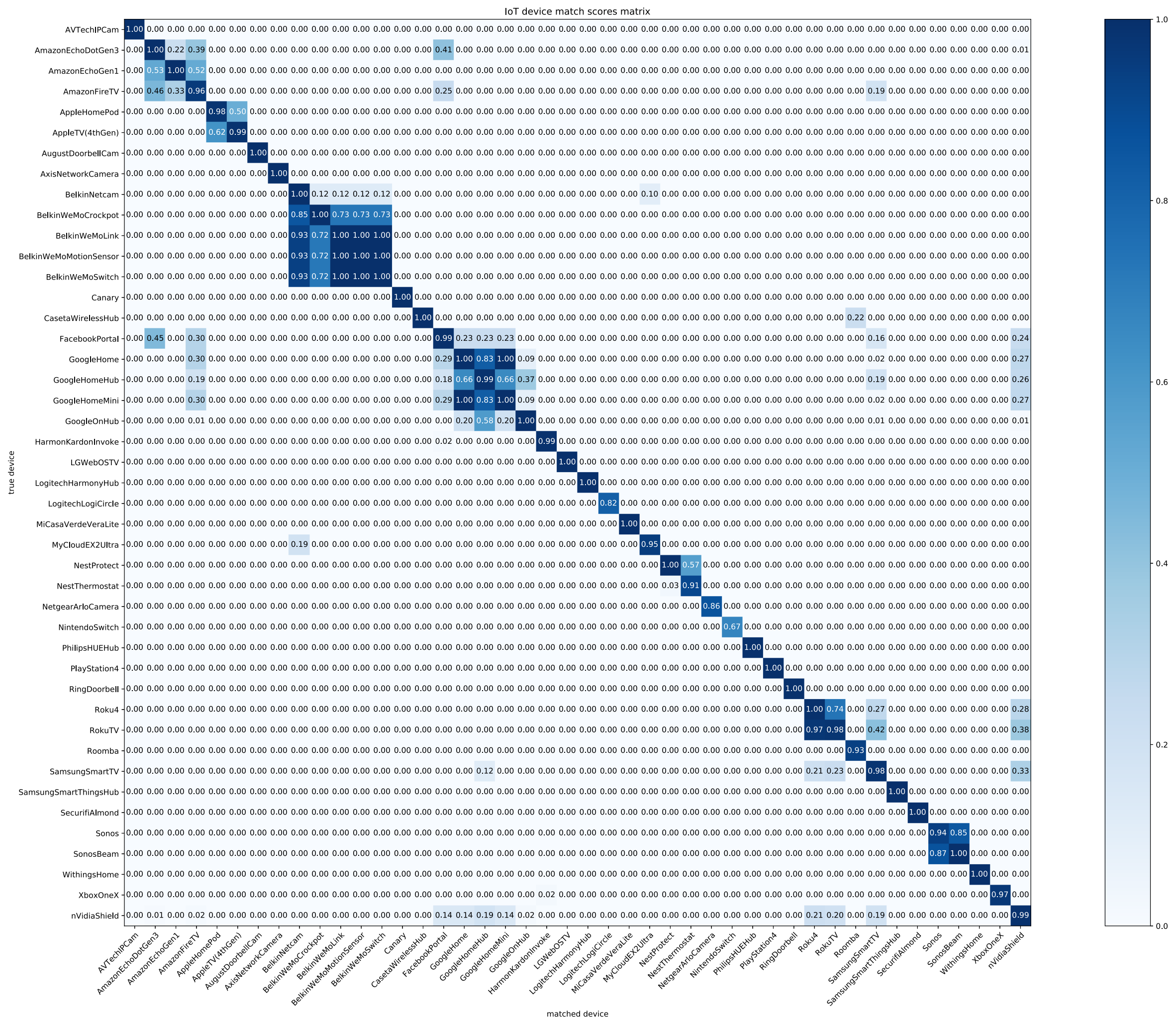


Figure 8: Confusion among IoT device traffic profiles

TABLE 2: Traffic models learned from Belkin devices

Domain \ Device	Link	MotionSensor	Switch	Netcam
api.xbcs.net	0.042	0.039	0.039	0.083
nat.xbcs.net	0.042	0.039	0.039	0.044
origin.belkin-us.tendplatform.com	–	–	–	0.083
time.windows.com	–	–	–	0.208
www.belkin.com	–	–	–	1.0

of traffic for computing the similarity scores. Given a device I_i and a test day t, let $s_t(k,i)$ be the similarity between P_k and traffic from I_i during t. Figure 8 reports the average similarity $\bar{s}(k,i) = \text{avg}_t\{s_t(k,i)\}$ for each pair (k,i).

Notice that, by construction, the IoT traffic models are not symmetric. Namely, given two models P_i and P_j respectively learned from devices I_i and I_j, in general $s_t(i,j) \neq s_t(j,i)$. The asymmetry is due to the projection of the vector representing the DNS traffic being tested upon in the vector space of the model for which similarity is being measured, as explained in Section 3.2 (this projection is important for enabling the detection of multiple devices even when they reside behind the same NAT).

From Figure 8, we can also notice that there is some "confusion" among devices from the same manufacturer. For instance, the statistical model for Belkin's WeMo Link has a 1.0 average similarity over traffic generated by Belkin WeMo Motion Sensor, and Belkin WeMo Switch, and a 0.93 average similarity over traffic from Belkin Netcam. We can see why from Table 2, which shows the statistical profile for each of the devices above.

For each device in Table 2, its column represents the set of domains queried by the device over the entire training period. Also, for each domain, it shows the probability that the domain will be queried in any time window of length w (one hour, in our experiments). Notice that symbol '$-$' indicates that the domain name was not queried by the device in the corresponding column. As can be seen, the Link, Motion Sensor, and Switch have an almost identical DNS profile (likely because they use the same firmware). On the other hand, the Netcam behaves somewhat differently. However, when projecting the set of domains queried by Netcam over the DNS query space of Link (i.e., keeping only the first two domain names in Table 2), for instance, we obtain a 2-dimensional vector with probabilities 0.083 and 0.044, which is fairly well aligned (though not perfectly) with the Link model, and thus yields a high cosine similarity of 0.93 shown in Figure 8 (notice that 0.93 is an average similarity across 14 experiments).

Similarly, Google Home and Google Home Mini appear to have exactly the same DNS behavior, which justifies their similarity values in Figure 8. After a quick investigation, we found that these two devices in fact run the exact same firmware version on slightly different hardware (specifically, both currently run *cast firmware* versions `1.42.180518`). On the other hand, Google Home Hub runs a different firmware version (`1.42.171872`), and its behavior is distinguishable (though still somewhat similar) from Google Home and Google Home Mini.

The results in the example above show that in some cases it is not possible to distinguish among certain devices from the same manufacturer using only DNS traffic, in that they have the exact same domain name query behavior. When one of those fingerprints matches, say for example the Belkin WeMo Switch fingerprint, `IoTFinder` will indicate that three devices match with similarity 1.0 (Switch itself, Link and Motion Sensor), one device matches with similarity 0.93 (Netcam), etc. In a real-world deployment scenario, this uncertainty means that a given client IP address may be hosting one or more of the Belkin devices that match with high similarity, but in this case we are not able to identify exactly what IoT device and how many are there (remember that we may be observing "mixed" traffic from a NAT). However, notice that this is not always the case for devices from the same vendor. For instance, Amazon, Logitech, and Nest devices appear to have a sufficiently different behavior, among products from the same vendor.

4.5. Resilience to DNS Packet Loss

Passive DNS data collection typically happens either from DNS resolver logs or via traffic sniffing. When collecting DNS messages from traffic, packet loss is almost unavoidable. Therefore, we also make an effort to analyze how resilient our IoT detection models are to noise due to packet loss. To this end, we simulate two types of loss, namely uniform and bursty, with different loss probabilities.

Due to space constraints, we discuss the details of this experiment in Appendix C, and visualize the results in Figure 9. In summary, the results show that while the partial AUC decreases as the packet loss probability increases, many IoT devices can still be detected with a high partial AUC even at fairly high loss rates (e.g., 50%).

4.6. Detecting IoT Devices Behind a NAT

In this section, we evaluate if the IoT devices can be detected when their traffic is mixed-in with many other IoT and non-IoT devices. To this end, we perform an experiment in which we take all DNS traffic related to one day of traces from our IoT lab, and rewrite all source IP addresses to a same single IP address, p (the exact IP address value itself does not influence detection). Similarly, we select 60 different devices from one day of our `LDNS` traffic, picking at random 10 IP addresses from each of the following labels: Windows, Mac, Linux, Android, iPhone, and iPads. We then also rewrite their source IP address as p. In other words, we pretend that there are more than 100 devices behind a NAT, and that therefore all these devices share the same IP address. Furthermore, we pretend that their DNS traffic is collected "above" the NAT, and the traffic is therefore all "mixed." We then match our previously trained IoT models to the dataset obtained as described above, to determine what devices we were able to detect, and what would be the level of false positives (FPs) and false negatives (FNs).

First, we compute the baseline: how many devices could we detect if the NAT was not present? To answer this question, we match our IoT fingerprints trained between August 4 and August 18, 2019, and match them against all IoT lab traffic from August 28, 2019 (10 days after training ended). On that test day, 52 IoT devices generated DNS traffic (one device was inactive). After computing the similarity between our IoT models and the test traces, we applied the automatically computed detection thresholds for two different maximum false positive

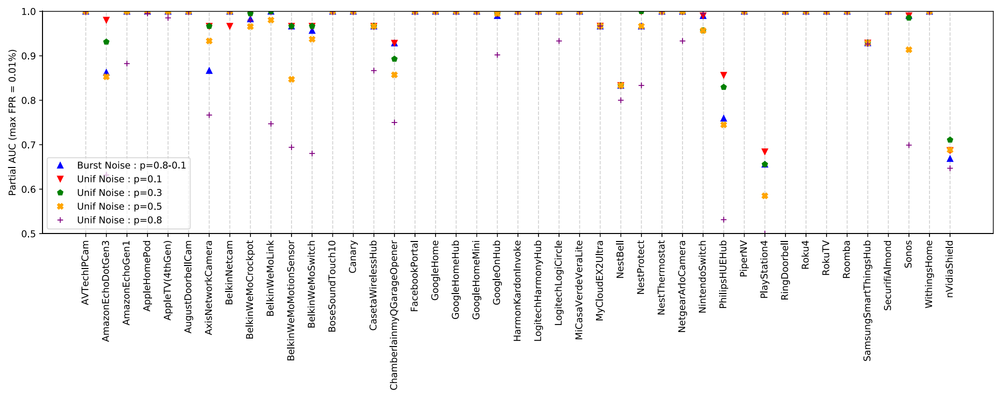


Figure 9: Fingerprints accuracy tested over noisy DNS traffic. DNS packet loss is simulated using two different models: uniform random loss, and bursty loss. We consider a single two-week training / test period. Packet loss is applied to the test dataset.

values: 0.1% and 0.01%. With these settings, we were able to detect 50 and 46 devices (i.e., 2 and 6 FNs), respectively (naturally, the lower the false positives we require, the lower the detection rate will tend to be).

As a second experiment, we "mixed" the traffic from all 52 IoT devices by rewriting their source IP addresses, as explained earlier. Essentially, in this setting we are simulating a NAT'ed network with 52 different IoT devices behind the same IP. Applying the same IoT models used above, and the same thresholds computed for maximum FP rates of 0.1% and 0.01%, we were able to detect 45 and 38 devices (i.e., 7 and 14 FNs), respectively. The decrease in the number of detected devices is due to the fact that mixing the traffic alters the frequency with which some domains appear to be queried by the NAT.

Third, we "mixed" the IoT traffic from all 52 IoT devices with traffic from the 60 non-IoT devices selected at random from 6 device types in `LDNS`, as mentioned earlier. As before, we match our IoT model against the (simulated) NATed traffic, with detection thresholds computed for a maximum FP rates of 0.1% and 0.01%. We were able to detect 40 and 30 IoT devices (i.e., 12 and 22 FNs), respectively.

Finally, we matched our IoT fingerprints against only non-IoT traffic from the 60 devices mentioned above, again pretending they were behind the same NAT, and we found no false positives for either of the two FP rates. These results show that, even in the challenging case of 112 different IoT and non-IoT devices behind a NAT, we were able to detect up to 40 out of 52 IoT devices with no false positive (the full list of 40 detected devices and 12 missed devices is reported in Appendix D).

4.7. Detecting Third-Party IoT Devices

We had the opportunity to test `IoTFinder` against network traffic collected at a separate lab network setup by an external, third-party entity that is completely unrelated to our organization. We refer to this dataset of network traces as `TPIoTDNS`. The dataset consists of 8 days of full DNS traffic traces. All DNS traffic had the same source IP address, which appears as a NAT device or DNS resolver that performs DNS caching. The network hosted a number of IoT devices, as well as other non-IoT devices, including a Raspberry Pi, a number of Windows and Linux virtual machines, etc. This network was setup by the third-party for a security-related exercise, and also hosted a number of malware infections that generated malicious (e.g., DGA or C&C-related) DNS queries.

After matching our models against the third-party traces, we computed the IoT device detections that exceeded the detection thresholds computed for 0.01% FPs, and then consulted the third-party data provider to verify what devices we correctly detected and what we might have missed. The third-party data provider confirmed that they had setup 6 active IoT devices (though not all devices were active on all 8 days): a MyCloud EX2 Ultra NAS device, a Roku streaming device, a Samsung SmartThings hub, an Apple TV box, a Netgear Arlo camera, and a Withings Home camera. Of these, we detected all devices correctly, except for Apple TV, which we consistently missed because the similarity scores generated for each day by our model were below the threshold computed for 0.01% FPs. We also had zero false positives.

We manually investigated why we missed the Apple TV device, and we found that while the detection threshold was automatically set to 0.763, the third-party device matched our model with a similarity score between 0.507 and 0.537. As demonstrated earlier (Section 4.6), some devices are more difficult to detect among "mixed" IoT and non-IoT traffic, and this may be the reason why our models did not match the third-party Apple TV device.

4.8. Estimating IoT Populations

We also deployed `IoTFinder` on DNS traffic collected at a large US ISP. While we do not have ground truth for the ISP traffic and cannot compute the accuracy of our device identification results, we believe these measurements are still useful, in that they aim to show that our DNS-based IoT identification system is highly efficient. To achieve this goal, we invested significant time in an Apache Spark-based implementation of the system (written in Scala) to leverage high levels of parallelism.

To concretely measure how efficiently we can match our IoT fingerprints, we use our IoT models trained over a period of two weeks, and test them against one full day of ISP DNS traffic. Our experiments are performed using a Spark cluster consisting of about 100 compute nodes, using 513 VCores and approximately 1.2TB of

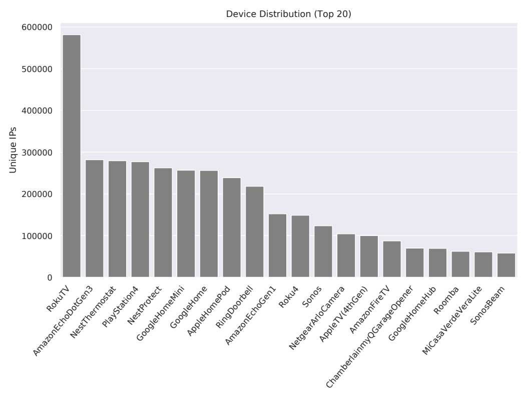


Figure 10: IoT device distribution in large US ISP network (top 20 devices by count).

total memory allocated to our job. With this setup, deploying our fingerprints over the DNS traffic of more than 40 million ISP clients takes about 72 minutes. Figure 10 shows the distribution of the top 20 most popular devices that matched our models.

5. Discussion and Limitations

Unlike network flows (e.g., TCP/UDP flow summaries and statistics), DNS traffic is typically more light weight and easier to collect from geographically distributed Internet vantage points. At the same time, the granularity of information contained in DNS messages is significantly lower than that available in detailed flow statistics (e.g., reported via Cisco NetFlow). Yet, our DNS-based IoT fingerprints are able to model IoT traffic behaviors and enable the accurate detection of many (though not all) IoT devices, even when they are co-hosted behind a NAT along with many non-IoT devices.

In some cases, DNS-based traffic models are unable to capture differences between separate devices produced by the same vendor. For instance, Google Home and Google Home Mini smart speakers have identical DNS behavior, as discussed in Section 4.4. This is why the IoT device population numbers reported in Figure 10 for those devices are the same, and should be interpreted as stating that either device is present in the network, with the reported count as an aggregate of the two. This and other similar cases can be inferred by analyzing the "confusion matrix" reported in Figure 8 (Section 4.4). At the same time, whenever an IoT fingerprint match is found to pass the related detection threshold, our system reports not only that a device has been detected, but also with what similarity. Therefore, when the similarity is not identical, it can be used as a sort of confidence score so that different devices can be disambiguated by selecting the device with the highest similarity.

Different devices that run similar software may also be a source of confusion, for instance in the case of devices that use the Alexa Voice Service (AVS) API or the Google Assistant API. However, in our experiment we did not find this to be an issue. For instance, some of the non-Amazon IoT devices in our lab are Alexa-enabled (e.g., SonosBeam). However, those devices can be distinguished well from Amazon devices such as AmazonEchoGen1 and AmazonEchoDotGen3. Also, our LDNS dataset includes traffic from more than 16,000 mobile devices, including iPhones and Android devices that may use the Amazon Alexa or Google Assistant app. Yet, we show that our system can accurately detect most IoT devices with very few false positives. The main reason is that while there may be common domains queried by different devices and apps, overall the combination of domains that are queried by a device and the (approximate) frequency with which they are queried make the device distinguishable.

Accurately detecting IoT devices becomes increasingly more difficult when a large number (e.g., hundreds) of IoT and non-IoT devices are hosted behind the same IP (e.g., behind the same NAT). Nonetheless, we showed that our statistical fingerprints are still able to correctly detect a large variety of IoT devices with no false positives, even in such challenging scenarios (see Section 4.6). It should be also noted that currently our system is not capable of identifying how many instances of the same device are behind a NAT. The main reason is that the DNS caching effects will likely "compress" query frequency information, making it difficult to idenitfy the exact number of devices that exhibit identical DNS behavior. However, it is worth noting that once we know that a specific IoT device model is behind a NAT, if that device presents known vulnerabilities then we can infer that the hosting network is potentially vulnerable to attacks, regardless of the exact number of vulnerable identical devices in it.

Naturally, our approach is limited to non-encrypted DNS traffic, since we need access to the domain name queried. Recently, there has been a push towards encrypting DNS traffic, using DoH (`RFC8484`) or DoT (`RFC7858`). In particular, *centralized DoH* may represent an issue, in that it may prevent local network operators (including ISPs) from gathering details about DNS traffic generated from their clients. However, we should consider a few observations. First, DoH is likely to have a negative impact on other existing security applications, and there are proposals that are being made to try to prevent this [22]. Secondly, DoH is currently limited mostly to DNS queries issued by browser. As such, if DoH becomes the default protocol for major browsers, IoT devices will likely be easier to detect, since their DNS queries will be mixed-in with a much smaller amount of non-DNS traffic. Future IoT devices may eventually start adopting DoH or DoT as well. However, for many legacy IoT devices whose resources are constrained and don't allow for adding DNS encryption software, their DNS traffic profile will likely become increasingly distinguishable. Finally, while new (or updated) IoT devices that adopt DoH may be hidden from direct ISP monitoring, our methodology remains valid, in that it could be applied by a DoH operator (e.g., Google or Cloudflare), instead of single ISPs, to estimate the population of IoT devices around the Internet.

6. Related Work

IoT devices have recently received considerable attention by cybersecurity researchers, partially due to their increasing popularity and their involvement in notable security incidents [5]. In particular, a number of works have explored using network traffic analysis to detect IoT devices in a network or study their behavior [23]–[37].

A number of these works leverage network flow statistics for device identification [23]–[25] or compromised device detection [26]–[28]. Other works have used an analysis of different application-layer protocols, such as HTTP, Telnet, DNS, etc., to identify IoT devices [29]–[32]. Furthermore, active probing techniques have also been proposed to identify IoT devices by scanning network assets [28], [33], or using the devices' MAC address to identify vendor and product type [38], [39]. It is worth noting that many of these works require access to the local network where the IoT devices are hosted, or to network flows, which are often expensive to collect at very large scale. On the other hand, we focus on efficiently detecting IoT devices based only on large-scale passive DNS traffic analysis, since globally distributed DNS traffic is typically easier to collect at scale (and is in some cases available for purchase).

A few recent works study the characteristics of DNS traffic generated by IoT devices [35], [36] and discuss related privacy implications [34], [40]. More recently, two works have considered using DNS traffic for IoT device detection [19], [37]. To the best of our knowledge, [19], [37] are the closest works to ours.

In [19], the authors use both domain name queries and destination IP addresses contacted by IoT devices to build a device detection system. As mentioned in Section 2.2, Guo et al. [19] proposed a number of manually devised heuristics to filter out IoT-queried domains that may raise false positives. For instance, given a set of domains queried by an IoT device, they probe the domains to determine whether they are "human facing" (i.e., whether they contain human-accessible content). Furthermore, they filter out domain names that do not appear to include the IoT device's vendor name. However, as discussed in Section 2.2, this heuristic may discard domain names that are instead highly discriminative and can be used to more accurately detect the presence of a specific IoT device. More importantly, [19] proposes to use single domain names for detection, which can cause significant confusion among devices that have overlapping DNS behavior (e.g., different device models from the same vendor). On the other hand, `IoTFinter` does not rely on the heuristics mentioned above, and can instead automatically learn IoT device detection models based on *combinations* of domain names queried by the devices. In addition, `IoTFinter` automatically learns a detection threshold for each statistical IoT fingerprint, so to limit possible false positives to a desired tolerable amount.

After building `IoTFinter`, we became aware of a recently published work [37] that also proposes the use of TF-IDF to model DNS traffic generated by IoT devices. However, [37] is quite different from our work. First, the system proposed in [37] requires that the traffic to be classified must be known to originate from a single IoT device. Quoting from [37]: "[the detection] algorithm should only be invoked when it is known that the traffic document belongs to a device that is known to be an IoT device." On the contrary, `IoTFinder` does not require such prior knowledge, and is able to detect IoT devices even when they are co-hosted behind the same IP address (e.g., behind a same NAT) along with many other IoT and non-IoT devices. Secondly, in [37] the detection thresholds must be set manually by an operator, whereas we devise an algorithm to label real-world non-IoT traffic and automate the process of learning a detection threshold per each device model to achieve a predetermined maximum tolerable false positive rate. Also, the detection pipeline presented in [37] relies on WHOIS records and x509 certificates to identify the vendor of an IoT device. However, in many practical cases the owner of the domain name may not be the same as the vendor, such as when private WHOIS records are used, or when services are hosted on a cloud-related domain (e.g., AWS). In this case, the device may ultimately be assigned to the wrong vendor. This erroneous decision would then cascade to the device type classifier, which relies on the success of the vendor classifier for correct device identification [37]. Our work is different, because we rely only on DNS traffic analysis, and make no assumptions on the ownership of the domain names queried by an IoT device. Furthermore, we focus on large-scale detection, and implement `IoTFinter` to efficiently detect many different IoT devices within large ISP networks hosting tens of millions of clients.

There also exists a proposal to enable the identification of IoT devices via automatic DNS name registrations [41]. However, it is unclear if this proposal will be widely adopted in the future, because it requires the collaboration of IoT vendors. Furthermore, legacy IoT devices may remain undetectable via this mechanism. On the other hand, our system does not require the collaboration of IoT vendors, and can detect IoT devices "as is", including legacy devices that may not support future protocols.

7. Conclusion

We presented `IoTFinder`, a system for efficient, large-scale *passive* identification of IoT devices. `IoTFinder` leverages distributed passive DNS data collection and a machine learning-based approach to accurately identify a large variety of IoT devices based solely on their *DNS fingerprints*. `IoTFinder` is also independent of whether the devices reside behind a NAT or other middleboxes. We evaluated `IoTFinder`'s accuracy in several different settings, including computing detection results over a third-party IoT traffic dataset and DNS traffic collected at a US-based ISP hosting more than 40 million clients. The experimental results showed that our approach allows for accurately detecting many diverse IoT devices, even when they are hosted behind a NAT and their traffic is "mixed" with traffic generated by other IoT and non-IoT devices hosted in the same local network.

Acknowledgments

We would like to thank Panagiotis Kintis for providing very helpful advice on our Spark/Scala system implementation and for his valuable research insights. We also thank the anonymous reviewers and shepherd for the constructive comments and suggestions on how to improve this paper. This material is based in part upon work supported by the Defense Advanced Research Agency (DARPA) under Contract No. 2106EHP. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[1] "IoT Growth," https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide.

[2] "Prediction for the growth of IoT devices," https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1093–1110.

[4] "VPNFilter," https://www.ic3.gov/media/2018/180525.aspx.

[5] "From homes to the office: Revisiting network security in the age of the iot," https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/from-homes-to-the-office-revisiting-network-security-in-the-age-of-the-iot.

[6] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by internet-wide scanning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 542–553.

[7] "Censys," https://censys.io.

[8] "Shodan," https://shodan.io.

[9] K. Borgolte, S. Hao, T. Fiebig, and G. Vigna, "Enumerating active ipv6 hosts for large-scale security scans via DNSSEC-signed reverse zones," in *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 438–452. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2018.00027

[10] A. Murdock, F. Li, P. Bramsen, Z. Durumeric, and V. Paxson, "Target generation for internet-wide ipv6 scanning," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17, 2017.

[11] P. Kintis, N. Miramirkhani, C. Lever, Y. Chen, R. Romero-Gómez, N. Pitropakis, N. Nikiforakis, and M. Antonakakis, "Hiding in plain sight: A longitudinal study of combosquatting abuse," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 569–586.

[12] C. Lever, R. Walls, Y. Nadji, D. Dagon, P. McDaniel, and M. Antonakakis, "Domain-z: 28 registrations later measuring the exploitation of residual trust in domains," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 691–706.

[13] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon, "Detecting malware domains at the upper DNS hierarchy." in *USENIX security symposium*, vol. 11, 2011, pp. 1–16.

[14] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS." in *USENIX security symposium*, 2010, pp. 273–290.

[15] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: detecting the rise of dga-based malware," in *21st USENIX Security Symposium*, 2012, pp. 491–506.

[16] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive DNS analysis." in *Ndss*, 2011, pp. 1–17.

[17] R. Perdisci, I. Corona, and G. Giacinto, "Early detection of malicious flux networks via large-scale passive DNS traffic analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 5, pp. 714–726, 2012.

[18] "FarSight DNS," https://www.farsightsecurity.com/solutions/dnsdb.

[19] H. Guo and J. Heidemann, "Ip-based iot device detection," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, ser. IoT S&P '18, 2018.

[20] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[21] S. D. Walter, "The partial area under the summary roc curve," *Statistics in Medicine*, vol. 24, no. 13, pp. 2025–2040, 2005.

[22] J. Livingood, M. Antonakakis, B. Sleigh, and A. Winfield, "Centralized dns over https (doh) implementation issues and risks," https://tools.ietf.org/html/draft-livingood-doh-implementation-risks-issues-04, 2019.

[23] M. R. Santos, R. M. Andrade, D. G. Gomes, and A. C. Callado, "An efficient approach for device identification and traffic classification in iot ecosystems," in *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2018, pp. 00 304–00 309.

[24] O. Salman, I. H. Elhajj, A. Chehab, and A. Kayssi, "A machine learning based framework for iot device identification and abnormal traffic detection," *Transactions on Emerging Telecommunications Technologies*, p. e3743, 2019.

[25] S. Marchal, M. Miettinen, T. D. Nguyen, A.-R. Sadeghi, and N. Asokan, "Audi: Toward autonomous iot device-type identification using periodic communication," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1402–1412, 2019.

[26] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for IoT," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 756–767.

[27] I. Alrashdi, A. Alqazzaz, E. Aloufi, R. Alharthi, M. Zohdy, and H. Ming, "Ad-iot: anomaly detection of iot cyberattacks in smart city using machine learning," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2019, pp. 0305–0310.

[28] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.

[29] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.

[30] 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 *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2017, pp. 559–564.

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

[32] B. A. Desai, D. M. Divakaran, I. Nevat, G. W. Peter, and M. Gurusamy, "A feature-ranking framework for iot device classification," in *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE, 2019, pp. 64–71.

[33] A. Dainotti, K. Benson, A. King, B. Huffaker, E. Glatz, X. Dimitropoulos, P. Richter, A. Finamore, and A. C. Snoeren, "Lost in space: improving inference of ipv4 address space utilization," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1862–1876, 2016.

[34] N. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic," *CoRR*, vol. abs/1705.06805, 2017. [Online]. Available: http://arxiv.org/abs/1705.06805

[35] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *IEEE S&P*, 2019, pp. 208–226.

[36] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach," in *Proceedings of the Internet Measurement Conference*. ACM, 2019, pp. 267–279.

[37] F. Le, J. Ortiz, D. Verma, and D. Kandlur, "Policy-based identification of iot devices vendor and type by DNS traffic analysis," in *Policy-Based Autonomic Data Governance*. Springer, 2019, pp. 180–201.

[38] J. Martin, E. Rye, and R. Beverly, "Decomposition of mac address structure for granular device inference," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 78–88.

[39] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: an analysis of IoT devices on home networks," in *28th USENIX Security Symposium*, 2019, pp. 1169–1185.

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

[41] J. Jeong, S. Lee, and J. Park, "DNS name autoconfiguration for internet of things devices," https://tools.ietf.org/id/draft-jeong-ipwave-iot-dns-autoconf-04.html, year = 2019, month = April.

[42] "PF_RING," https://www.ntop.org/products/packet-capture/pf_ring.

[43] L. Rizzo, "Netmap: a novel framework for fast packet i/o," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 101–112.

Appendix A. User-Agent String Analysis

To label non-IoT devices in real traffic, we rely on `User-Agent` strings found in HTTP traffic. For instance, consider the strings in Example 1. Looking closely, it's easy to see that they in some cases reveal the type of hardware (e.g., `iPhone`, `PC desktop`, etc.) and/or the operating system, from which the hardware can be reasonably inferred (e.g., `Intel Mac OS X 10_10_5` likely indicates an Apple laptop or desktop system). To use this information, we collected similar strings from traffic traces collected withing our campus network for a few days. We then manually extracted a number of regular expressions that capture the main keywords that allow for inferring a device label for the vast majority of clients. Each `User-Agent` regular expression we extract corresponds to one of these 6 labels: Windows, Mac, Linux, Android, iPhone, and iPad. Devices that cannot be associated to one of those labels are discarded from further consideration. When a client issues an HTTP request, we extract its `User-Agent` string, match it to our set of regular expressions, and attribute the device label associated with the regular expression that matches it (if any).

Appendix B. Additional ROC Analysis

Figure 11 shows results for an experiment similar to what reported in Figure 7, with the only difference being that each test period is 4 days long, instead of 2 weeks long. As discussed in Section 4.3 the results show that the pAUC is not significantly different between the two experiments, which suggests that our IoT fingerprints do not need to be re-trained very frequently.

Appendix C. Resilience to DNS Packet Loss

One of the main challenges when collecting network traffic in large networks, including DNS traffic, is limiting the packet loss rate. While a number of effective approaches have been proposed to reduce packet loss (e.g., in [42], [43]), it may not always be possible to completely eliminate it. The effect packet loss has on a device's traffic profile is obvious: certain domain names that the device queried may be observed with lower frequency, or not observed at all (e.g., in case of domains that are infrequently queried to begin with, and are unluckily affected by packet loss). We therefore decided to evaluate how resilient our IoT models are to DNS packet loss. To this end, we simulate two packet loss scenarios:

- *Uniform loss*: each packet has the same chance p_u to be dropped.
- *Bursty loss*: packets have an increased chance p_b to be lost during certain time windows, compared to the loss probability p_n (with $p_n < p_b$) experienced at "normal" times.

While we acknowledge that the heuristics outlined do not make use of real-world packet loss model, they allow us to approximately quantify the resilience of our fingerprints to different types and levels of noise. We leave the evaluation of more realistic packet loss models to future work.

The bursty loss scenario is intend to (very coarsely) simulate the effect of traffic peaks at certain hours of the day, when packet loss may consequently worsen. In our case, we set the burst window duration to be one hour, and the time between two bursts to be approximately 2.5 hours, giving us 6 traffic bursts in 24 hours. For the uniform loss scenario, we vary p_u between 0.1 and 0.8, whereas for the bursty loss scenario, we fix $p_b = 0.8$ and $p_n = 0.1$.

After imposing the simulated traffic loss over our `IoTDNS` dataset, we tested our IoT fingerprints in a way similar to the experiments presented in Section 4.3, and obtained the results summarized in Figure 9 (notice that at the time when this experiment was run, some of the IoT devices in our lab were not active, and are therefore not represented in the figure). Naturally, in the case of uniform noise, the higher the packet loss, the more difficult it becomes to detect a device. However, even at $p_u = 0.8$ many of the devices can still be accurately detected. This may be due to the fact that if a device queries a set of domains relatively frequently, the noise has the effect of reducing the query counts by the same proportion. In this case, the cosine similarity may not be affected, because it would still measure the same angle between two vectors, even if one of the vectors has a smaller norm. Conversely, the bursty noise affects the traffic only during certain time windows, and queries during low traffic volume times will not be heavily affected. This gives us a chance to still detect the devices based on their activities outside of peak times.

Appendix D. List of Detected and Missed Devices in NAT experiment

Below, we report the list of IoT devices that we detected and we missed during the simulated NAT experiments in which all 52 active IoT devices are mixed-in, along with non-IoT devices, and the maximum FP rate is set to 0.1% (see Section 4.6):

```
=== DETECTED ===
AVTechIPCam
AmazonEchoDotGen3
AppleHomePod
AugustDoorbellCam
AxisNetworkCamera
BelkinWeMoCrockpot
BelkinWeMoLink
Canary
ChamberlainmyQGarageOpener
```

Example 1 Example `User-Agent` strings extracted from real-world HTTP traffic.

```
  1) Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0) Gecko/20100101 Firefox/67.0
  2) Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/603.3.8 (KHTML, like Gecko) [...] Safari/603.3.8
  3) Mozilla/5.0 (iPad; U; CPU OS 3_2_1 like Mac OS X; en-us) AppleWebKit/531.21.10 (KHTML, like Gecko) Mobile/7B405
  4) Mozilla/5.0 (iPhone; CPU iPhone OS 12_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148
  5) Spotify/8.5.31 Android/28 (SM-G960U)
  6) Spotify/110500153 Linux/0 (PC desktop)
```

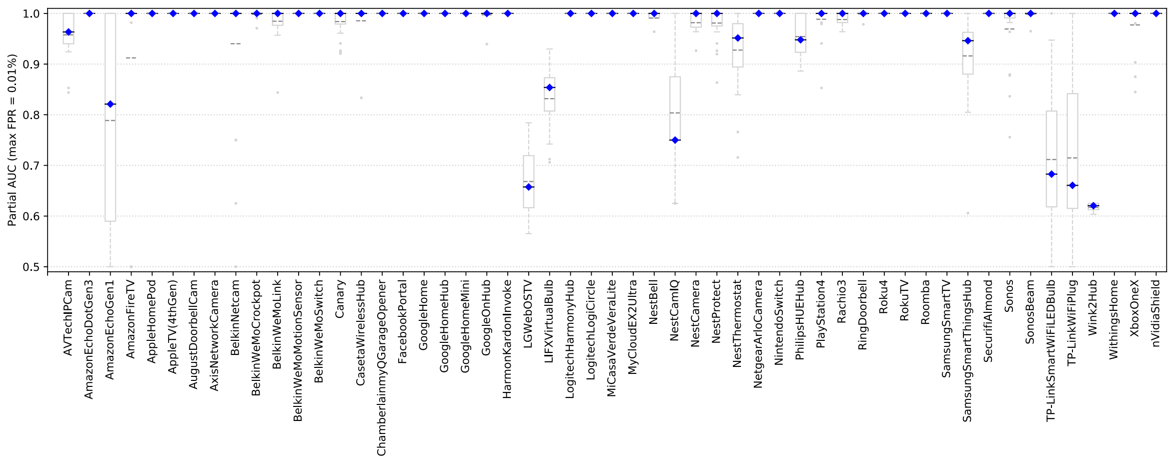


Figure 11: Fingerprints' accuracy: two weeks training / four days testing. Each fingerprint is trained over two weeks of IoT traffic and tested over traffic collected in the following two weeks. Each test period consists of a different two weeks training / four days testing period. Boxplots summarize the pAUC distribution over 20 different training/testing periods. Blue diamonds indicate the median pAUC.

```
FacebookPortal
GoogleHomeHub
GoogleOnHub
HarmonKardonInvoke
LGWebOSTV
LIFXVirtualBulb
LogitechHarmonyHub
LogitechLogiCircle
MiCasaVerdeVeraLite
MyCloudEX2Ultra
NestCamIQ
NestCamera
NestProtect
NestThermostat
NetgearArloCamera
NintendoSwitch
PhilipsHUEHub
PlayStation4
Rachio3
RingDoorbell
Roku4
RokuTV
Roomba
SamsungSmartThingsHub
SecurifiAlmond
Sonos
SonosBeam
TP-LinkSmartWiFiLEDBulb
TP-LinkWiFiPlug
WithingsHome
nVidiaShield

=== MISSED ===
AmazonEchoGen1
AmazonFireTV
AppleTV(4thGen)
BelkinNetcam
BelkinWeMoMotionSensor
BelkinWeMoSwitch
CasetaWirelessHub
GoogleHome
GoogleHomeMini
XboxOneX
```