# Backend-Service Fingerprints: Lightweight and Interpretable Identification of Household IoTs

Shayan Azizi*, Norihiro Okui†, Masataka Nakahara†, Ayumu Kubota† and Hassan Habibi Gharakheili*
*School of Electrical Engineering & Telecommunications, UNSW Sydney, Australia. Emails: {*s.azizi, h.habibi*}*@unsw.edu.au*
†KDDI Research, Japan. Emails: {*no-okui, ms-nakahara, ay-kubota*}*@kddi.com*

***Abstract*—The rapid proliferation of vulnerable Internet-of-Things (IoT) devices in home networks has prompted Internet Service Providers (ISPs) to identify and classify devices using passive traffic measurements. Existing approaches typically rely either on machine-learning models trained on fine-grained traffic features or on endpoint-based profiling (*e.g.,* domains or IP addresses), both of which face practical limitations in terms of cost, robustness, and interpretability.**

**We propose a light and interpretable fingerprinting approach based on a traffic attribute readily observable from any flow-level measurement: the backend network service (*e.g.,* TCP/443, UDP/53). Our key insight is to decouple *what* services define a device from *how actively* those services are used over time. Specifically, each device is associated with a single *service vocabulary* of essential services and with one or more *service-usage fingerprints* that capture stable activity modes. We: (1) develop a persistence-based method to infer essential backend services from longitudinal passive flow measurements, (2) introduce a generalized persistence representation for capturing service activity levels and an adaptive exporter that yields multiple service-usage fingerprints per device, and (3) evaluate a two-stage identification pipeline (vocabulary-based confinement followed by fingerprint matching) on 50 million IPFIX flows from 60 device types collected over multiple years. We publicly release our datasets. Our results show that backend-service fingerprints are interpretable, robust, and effective for IoT identification.**

***Index Terms*—Internet of Things, device fingerprinting, network traffic measurement, flow telemetry, service-level analysis**

## I. Introduction

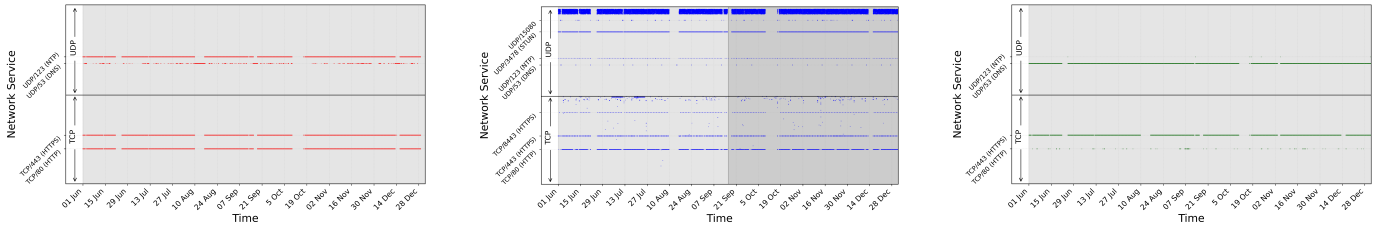
The rapid proliferation of Internet-of-Things (IoT) devices in residential networks has created new security and operational challenges [1] for Internet Service Providers (ISPs). Many IoT devices run poorly maintained firmware and rely on proprietary cloud backends, making it difficult for network operators to understand which devices are present and enforce appropriate access policies. As a result, identifying and classifying IoT devices using passive traffic measurements has become an important capability for ISPs and home gateways. From an operational perspective, such identification methods must be lightweight, interpretable, and robust to encryption and backend churn, as they are expected to operate continuously on large volumes of traffic at network edges. Existing approaches to IoT device identification typically rely on machine-learning (ML) classifiers trained on statistical traffic features [2]–[7] or on profiling the domain names contacted by devices [8]–[11]. ML-based methods can achieve high accuracy but often require costly feature extraction and produce opaque inferences, while domain-based approaches are increasingly hindered by DNS encryption, endpoint dynamism, and shared cloud infrastructure. These limitations motivate the search for alternative traffic abstractions that are robust, scalable, and amenable to operational interpretation.

978-1-6654-XXXX-X/XX/$31.00 ©2026 IEEE

In this paper, we focus on *backend network services* (§IV), which we define as protocol-port combinations (*e.g.,* TCP/443 or UDP/53) that devices use to communicate with their backends or shared backend infrastructures, and use them as a basis for fingerprinting. Backend services are readily available in standard flow-level telemetry, independent of payload encryption, and closely tied to device functionality. Compared to packet-level statistics or endpoint-based identifiers (*e.g.,* domains or IP addresses), backend services occupy a practical middle ground: they are stable enough to reflect device behavior while remaining observable using shallow telemetry such as standard IPFIX records.

However, a key challenge lies in identifying which observed services are *essential* to a device. Although devices that rely exclusively on fixed client–server ports are straightforward, many devices exhibit more complex behaviors such as peer-to-peer or rendezvous communication, which introduces ephemeral or peer-selected ports that are unstable and unsuitable for fingerprinting. Moreover, even after isolating essential services, service lists alone are often insufficient for unique identification because many devices share common backend services.

This paper makes three contributions. **(1)** We introduce the notion of a *service vocabulary* and develop a persistence-based method to infer essential backend services from longitudinal passive flow measurements (§V). **(2)** We show that while each IoT device is characterized by a single service vocabulary, it can exhibit multiple stable modes of service activity levels, and we develop a generalized persistence-based representation together with an adaptive exporter to capture one or more *service usage fingerprints* per device over time (§VI). **(3)** We design and evaluate a two-stage device identification scheme that combines vocabulary-based confinement with service-usage fingerprint matching, and validate it on two datasets consisting of 50 million IPFIX flows from 60 IoT device types collected over 2.5 years (§VII). Our datasets (§III) will be publicly released after this paper is de-anonymized.




(a) Amazon Echo. (b) I-O DATA QWatch Camera. (c) Line Clova Wave.

Fig. 1: Backend service usage over the training period of IoT-ServTrace-2019 for three IoT devices, illustrating clear differences in the composition and stability of their contacted services.

## II. Related Work

**IoT fingerprinting based on traffic features.** A large body of prior work uses machine-learning models trained on packet-level or flow-level traffic features to identify IoT devices [2]–[7]. However, they often rely on computationally expensive features [12] and produce opaque inferences that are difficult to interpret or validate in operational environments [13]. In addition, the performance of ML classifiers is shown to degrade within few-week timeframes [14], [15] due to evolving traffic patterns.

**Endpoint and DNS-based fingerprinting.** An alternative class of approaches fingerprint devices by profiling contacted endpoints, such as domain names or IP addresses [10], [11], [16]. Although endpoint-based fingerprints are lightweight and interpretable, they are fundamentally challenged by many-to-one and one-to-many mappings, provider-side IP churn, and the growing deployment of encrypted DNS. These limitations are becoming more pronounced with the increasing use of shared cloud infrastructures by IoT backends [9], [17].

**Backend and service-level analysis.** Several recent measurement studies have examined the backend infrastructures and service dependencies of IoT, providing information on cloud reliance, geolocation effects, and backend heterogeneity [9], [18], [19]. These works characterize where and how IoT devices communicate, but do not address how backend service information can be leveraged for robust device fingerprinting. Related efforts classify destinations as functionally essential or non-essential for traffic filtering using supervised learning at the destination level [20]; however, these approaches focus on traffic control rather than service stability or device identity.

**Research gap.** In summary, existing approaches either rely on unstable endpoints or on opaque ML models that require costly traffic features, and none provide a way to extract essential backend services or to reason about how those services are used over time. Our work addresses this gap by developing a passive fingerprinting method that separates the composition of the backend service from the behavior of service use, enabling robust and interpretable identification of IoT devices based on representations at the service-level.

## III. Our IoT Traffic Dataset

**Data Collection Setup:** We use IP Flow Information Export (IPFIX) records collected by an ISP from a controlled lab testbed across two periods: mid-2019 to the end of 2020 (hereafter *IoT-ServTrace-2019*) and the full year of 2022 (*IoT-ServTrace-2022*). We use 1 Jun–31 Dec 2019 for training and 2020 for testing in IoT-ServTrace-2019; for IoT-ServTrace-2022, we use 1 Jan–30 Jun for training and the remainder for testing. The datasets include traffic from 18 and 52 IoT device types (make-and-model), respectively, each active throughout its capture period. Each IPFIX record includes standard header-derived identifiers and fields such as MAC and IP addresses, transport-layer protocols, and port numbers. The datasets are collected and processed in accordance with institutional ethical guidelines.

**Data Preparation:** A logical flow may be exported as multiple IPFIX records due to timeouts or delayed packets. Since our analysis relies on the usage of temporal services, we merge redundant records using flow identifiers and temporal proximity. We discard local and IPv6 traffic, retaining only external IPv4 flows. After preprocessing, the combined datasets comprise approximately 50 million IPFIX records from 60 device types. The service traces are made public [21].

## IV. Motivating Examples

We illustrate why backend services are informative for fingerprinting and why isolating *essential* services is necessary. Fig. 1 shows service usage traces for three devices in 2019: Amazon Echo (2nd Gen), I-O DATA QWatch Camera, and Line Clova Wave. Each point denotes the use of a destination service (port) over time; dense regions indicate frequent activity.

Amazon Echo and Line Clova Wave rely on a small, stable set of services, including TCP/80 (likely HTTP), TCP/443 (likely HTTPS), UDP/53 (likely DNS), UDP/123 (likely NTP). In contrast, the QWatch Camera exhibits three groups: (i) **standard** services (*e.g.,* TCP/443, TCP/8443, UDP/53, UDP/123, UDP/3478), (ii) **proprietary** services (*e.g.,* UDP/15080 used by I-O DATA's NAS solutions [22]), and (iii) **sporadic** ports spanning wide ranges (*e.g.,* TCP/UP ports between 32000 and 65000, likely NATed port numbers of remote peers connecting to the camera for streaming via hole-punching). The sporadic services correspond to unstable behaviors and are therefore unsuitable for fingerprinting.

Fig. 2 illustrates this behavior by showing a truncated range of UDP ports (55600—55620) used over two sub-windows: 1 Jun-16 Sep (lighter shade) and 17 Sep-31 Dec (darker

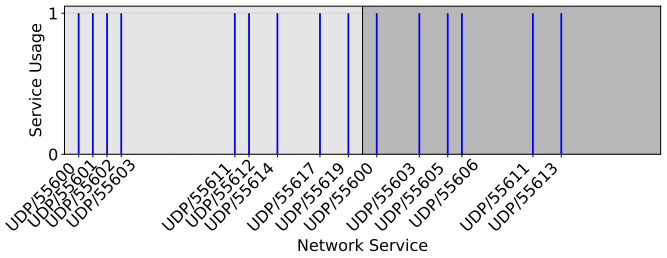
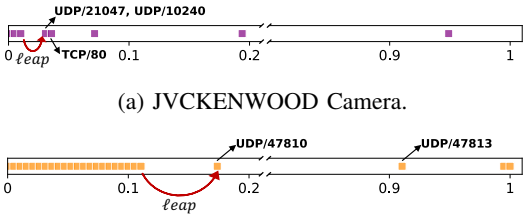



Fig. 2: Truncated set of services used by I-O DATA Camera across two consecutive sub-windows, highlighting non-stationary and non-overlapping port usage.




(a) JVCKENWOOD Camera.

(b) Panasonic Doorphone.

Fig. 3: Persistence values of all services used by (a) JVCKENWOOD Camera and (b) Panasonic Doorphone. Each device exhibits a dense cluster of low-persistence services followed by a clear leap, marking the transition from non-essential to essential services.

shade), corresponding to the periods highlighted in Fig. 1b. A value of 1 in the plot indicates that the service is used within the sub-window; 0 otherwise. Even over periods longer than 15 weeks, the sets of contacted ports differ substantially. Over shorter time scales, this non-stationarity is even more pronounced. Treating such ports as defining characteristics would introduce noise. We therefore define a device's *service vocabulary* (essential services) as the union of its standard and proprietary services.

However, service lists alone are insufficient when devices share same services, *e.g.,* Amazon Echo and Line Clova Wave. These devices can be distinguished by differences in the *activity levels* of services, motivating a second dimension: *service usage fingerprinting*. These observations motivate our two-step approach: extract a stable service vocabulary (§V) and then use fingerprinted usage patterns on that vocabulary (§VI).

## V. Constructing and Utilizing Service Vocabularies

An effective device fingerprint begins with identifying the backend services that are essential to the device's operation. In this section, we describe how to infer the *service vocabulary* of a device (*i.e.,* the set of essential backend services) from passive flow measurements and how this vocabulary can be used to confine candidate device classes during identification.

### A. Inferring Essential Backend Services

Given a longitudinal trace of flow records for a device (*e.g.,* Fig. 1), our goal is to separate essential services from non-essential ones. Essential services correspond to stable backend functionality, whereas non-essential services arise from incidental or unstable behaviors (*e.g.,* NAT hole-punching) and should not contribute to fingerprinting.

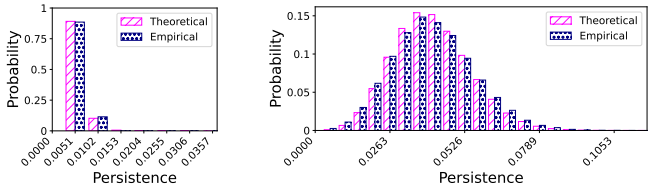
Naïvely relying on protocol–port mappings (*e.g.,* HTTP↔TCP/80) are insufficient in practice. Firstly, operating systems and network stacks may temporarily re-purpose reserved ports. For example, we observed a single TCP/20 flow in the I-O DATA Camera trace, a port typically associated with File Transfer Protocol (FTP) data transfer; however, no TCP/21 control channel was present, and the session was initiated remotely. Conversely, some proprietary backend services consistently use non-standard ports that are indistinguishable from transient ones when viewed in isolation and would be incorrectly discarded as non-essential if one relied solely on conventional protocol–port mappings. These observations highlight the need for a more robust criterion to identify essential backend services.

Since we assume access to flow traces over extended periods, one possibility is to identify essential services using the relative frequency of flows per port. However, the raw frequency can be misleading. For example, UDP/53 constitutes only $0.085\%$ of all flows for the I-O DATA Camera in 2019; yet the device uses it on $11\%$ of active days. This shows that ***persistence*** (measured by the fraction of active days on which a service appears at least once) is a naturally more reliable indicator of essentiality than raw flow counts.

For a device active on $n_a$ days, persistence values lie on a lattice with spacing $1/n_a$. Empirically, across devices, we observe that non-essential services occupy a continuous cluster of low persistence values, followed by a *discontinuity* (a leap larger than $1/n_a$), after which essential services appear. It is important to note that the location of this discontinuity is device-specific, making fixed global thresholds ineffective.

Figures 3a and 3b illustrate this behavior for two representative devices. For the JVCKENWOOD Camera, persistence values corresponding to numerous UDP ports populate consecutive low levels before a leap separates a small set of services (*e.g.,* UDP/21047, UDP/10240, and TCP/80) that remain active for many days. For the Panasonic Doorphone, a similar dense cluster of low-persistence services is followed by a distinct jump corresponding to UDP/47810 and related ports, which are consistently present throughout the activity period of the device. In both cases, we have confirmed that the services beyond the leap correspond to stable backend behavior, while those before the leap reflect unstable or incidental communication.

**Patterns in persistence values.** Starting from zero, we identify the first persistence leap of size at least $2/n_a$ and classify all subsequent services as essential. To assess the soundness of this criterion, we pose the following question: *(Q1) What is the probability that a non-essential service attains the persistence value observed immediately after the discontinuity?* Also, to further justify that the lowest persistence of $1/n_a$ is reliably populated by non-essential services, and thus discontinuity




(a) JVCKENWOOD. (b) Panasonic Doorphone.

Fig. 4: Comparison of theoretical and empirical persistence distributions for non-essential services for: (a) JVCKENWOOD Camera, and (b) Panasonic Doorphone. The close alignment between the zero-truncated binomial model and observed prevalence values validates our statistical modeling of non-essential services and underpins the confidence guarantees used to assess errors in essential-service extraction (§V-A).

detection can begin from zero, we pose a second question: *(Q2) What is the probability that no non-essential service appears at persistence* $1/n_a$*?*

To address these questions, we model the distribution of the number of days on which a given non-essential service appears, denoted by $K$, where the corresponding persistence is given by $K/n_a$. Consider a device that generates $n$ non-essential flows over $n_a$ days using $m$ distinct non-essential services. Because we model the distribution conditioned on services being observed, each service contributes at least one flow deterministically, accounting for $m$ of the $n$ flows. Assuming that a given service appears once more on an arbitrary day with probability $p$, independently across days, we obtain $K-1 \sim \text{Binomial}(n_a, p)$. To express $p$ in terms of $m$, $n$ and $n_a$, we compute the probability $\pi$ that a given non-essential service does not appear again among the remaining $n-m$ flows in two ways. First, from the binomial model, $\pi = (1-p)^{n_a}$. Second, assuming that the remaining flows select non-essential services independently and uniformly at random from the $m$ services, we have $\pi = (1-1/m)^{n-m}$. Equating these expressions yields $p = 1-(1-1/m)^{\frac{n-m}{n_a}}$.

Fig. 4 compares the empirical distribution of persistence values for non-essential services with the corresponding binomial distribution predicted by the model above. For both representative devices, the theoretical and empirical distributions closely match, validating the suitability of the binomial model for characterizing non-essential service usage.

Using this model, we compute the probability that a non-essential service attains the persistence observed immediately after the discontinuity. For the JVCKENWOOD Camera and the Panasonic Doorphone, these probabilities are $1.41 \times 10^{-7}$ and $4.08 \times 10^{-13}$, respectively, indicating an extremely low likelihood of misclassifying a non-essential service as essential (Q1 above).

To address (Q2), we estimate the probability that no non-essential service appears at persistence $1/n_a$. With $m$ non-essential services observed, this probability is $(1-\Pr[K=1])^m$, which is evaluated as $7.15 \times 10^{-161}$ and $1.02 \times 10^{-26}$ for the JVCKENWOOD Camera and Panasonic Doorphone, respectively. Intuitively, this probability is negligible because non-essential services are numerous and their appearances are dispersed across days; as a result, at least one service almost surely appears on exactly one day.

It is worth noting that our theoretical analysis is not intended to directly detect non-essential services, but rather to quantitatively validate the soundness of the proposed persistence-discontinuity criterion for identifying essential services. Applying this persistence-discontinuity method to both datasets yields compact and interpretable service vocabularies, typically fewer than 10 services per device. Larger vocabularies arise only when devices intentionally use broad port ranges. Because vocabularies consist of explicit backend services, they are human-readable, stable over time, and well-suited for downstream identification.

### B. Vocabulary-Based Candidate Confinement

Service vocabularies enable an effective first stage of identification by confining the set of plausible device classes. Given test-phase traffic, we match observed services with vocabularies to eliminate incompatible devices. Because short windows may contain transient, non-essential ports, we sort observed services by frequency and iteratively filter candidates based on vocabulary membership, excluding services absent from all vocabularies, which are indicative of non-essential usage. This procedure substantially reduces ambiguity while avoiding premature elimination of the correct class and prepares the remaining candidates for service-usage fingerprint matching.

## VI. Service Usage Fingerprints

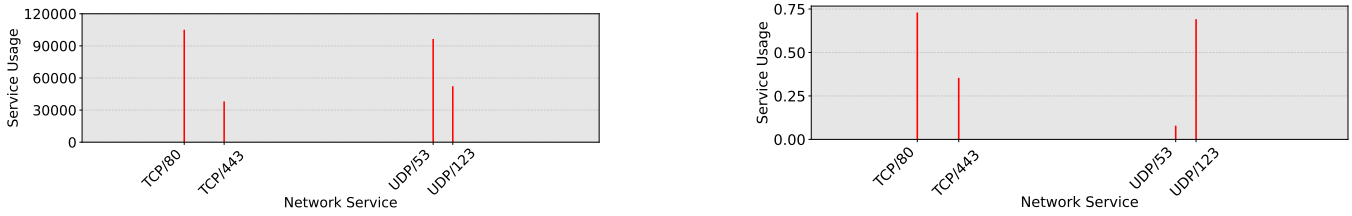
While a service vocabulary captures *what* backend services define a device, it does not describe *how* those services are used. As shown in §IV, devices that share identical service vocabularies may still exhibit distinct activity patterns. We therefore introduce *service usage fingerprints* to capture stable modes of service usage over time. Conceptually, each IoT device type is characterized by a single service vocabulary but may exhibit multiple service usage fingerprints, reflecting different but stable behavioral modes.

### A. Representing Service Usage

Given a window $W$ of flow records, we seek a representation that captures the relative activity of each service found in the device's service vocabulary. A natural baseline is to count the number of flows per service within $W$, which we refer to as the *Service Prevalence (SP)* representation. However, SP is overly sensitive to transient traffic fluctuations.

Fig. 5 illustrates this limitation using Amazon Echo. Some short-lived surges in DNS service activity inflate the apparent importance of UDP/53 under SP (Fig. 5a), even though UDP/53 exhibits the lowest sustained activity throughout the trace. Such transient effects distort the fingerprint's stability.

To address this, we introduce a persistence-based *Generalized (G) representation*. We partition a window $W$ into $g$ equal-sized sub-windows of duration $s$. For each service, we count the fraction of sub-windows in which the service appears at least once; dividing by $g$ yields an activity persistence




(a) SP-representation.

(b) G-representation.

Fig. 5: Comparison of (a) SP-representation versus (b) G-representation, for Amazon Echo. The SP-representation is distorted by a brief surge in UDP/53 (DNS) activity, whereas the persistence-based G-representation captures the device's stable service usage pattern.

**Algorithm 1** Service Usage Fingerprint Exporter.

**Require:** Anchor time $t_0$; initial window length $L_0$; maximum iterations $i_{\max}$; similarity threshold $\theta$; minimum flow-count growth rate $\delta$

**Ensure:** Fingerprint $\mathbf{r}_G(W_i; s)$ or *Did not converge*

$W_0 \leftarrow$ flow records in interval $[t_0,\, t_0 + L_0]$
$n_{\text{ref}} \leftarrow n(W_0)$
$\mathbf{r}_{\text{ref}} \leftarrow \mathbf{r}_G(W_0; s)$
**for** $i = 1$ to $i_{\max}$ **do**
  $W_i \leftarrow$ flow records in interval $[t_0,\, t_0 + 2^i L_0]$
  $n_i \leftarrow n(W_i)$
  **if** $n_i - n_{\text{ref}} > \delta \cdot n_{\text{ref}}$ **then**
    $\mathbf{r}_i \leftarrow \mathbf{r}_G(W_i; s)$
    **if** $\|\mathbf{r}_{\text{ref}}\|_1 > 0$ **then** ▷ Skip empty $W_0$
      **if** $s(\mathbf{r}_i, \mathbf{r}_{\text{ref}}) > \theta$ **then** ▷ Similarity check
        **return** $\mathbf{r}_i$
      **end if**
    **end if**
    $\mathbf{r}_{\text{ref}} \leftarrow \mathbf{r}_i$
    $n_{\text{ref}} \leftarrow n_i$
  **else**
    **continue** ▷ Insufficient new data
  **end if**
**end for**
**return** *Did not converge*

in $[0, 1]$, as shown in Fig. 5b. This representation, denoted $\mathbf{r}_G(W; s)$, suppresses short bursts (like the UDP/53 spike) while preserving sustained usage patterns. For sufficiently small $s$, the G-representation converges to SP; for moderate $s$ (*e.g.,* 5-minute sub-windows used in our evaluation), the G-representation yields greater robustness to transient anomalies.

### *B. Exporting Service Usage Fingerprints*

A key question is *how much traffic is required to export a reliable service-usage fingerprint*. Note that computing a single fingerprint from the entire training period would obscure distinct service-usage manifestations; in practice, devices often exhibit multiple stable usage modes over time, driven by factors such as firmware updates, configuration changes, user interaction patterns, or backend service evolution. Aggregating traffic over the entire training period would conflate these distinct modes into a single representation, reducing discriminative power. This motivates an adaptive mechanism that identifies and exports multiple fingerprints, each corresponding to a locally stationary usage pattern.

**Search Procedure.** We design an adaptive fingerprint exporter that searches for *locally stationary* usage patterns. Starting from an anchor time, the exporter computes $\mathbf{r}_G(W; s)$ over progressively larger windows using exponential growth (window lengths $L_0, 2L_0, 4L_0, \ldots$). As a stationarity criterion, at each step, the current representation is compared with a reference representation $\mathbf{r}_{\text{ref}}$ using cosine similarity, where $\mathbf{r}_{\text{ref}}$ is initialized as $\mathbf{r}_G(W_0; s)$ corresponding to the initial window $W_0$. When similarity exceeds a threshold $\theta$, the representation is exported as a fingerprint; otherwise, the reference is updated, and the search continues. If the relative increase in flow count is insufficient (inactivity or outages of the device), the current window is skipped. The maximum window size is capped at 64 days (approximately two months); if no fingerprint is exported within this budget, the exporter restarts the search from a later anchor time. In practice, most fingerprints are exported within 16 days, indicating that device usage stabilizes on relatively short time scales. We use a sub-window size of $s = 5$ minutes and a similarity threshold of $\theta = 0.95$. A formal description of the exporter is provided in Algorithm 1.

**Monitoring and Re-triggering.** After exporting a fingerprint, the system continuously monitors subsequent traffic to detect emerging usage patterns. We compare consecutive, equal-length validation windows against all previously exported fingerprints; if none exceeds the similarity threshold $\theta$, the exporter (Algorithm 1) is re-triggered to search for a new locally stationary mode. After exporting an additional fingerprint, validation resumes from the timestamp at which that fingerprint was exported. This mechanism allows the fingerprint set to adapt over time while remaining bounded, and enables incremental processing based on newly observed traffic. Excessively large validation windows may obscure local changes, whereas overly small windows do not provide stable representations for comparison against existing fingerprints. In §VII, we will evaluate validation windows of 8 days, 1 day, 12 hours, and 3 hours.

### *C. Matching Test-Phase Traffic*

During identification, we compute $\mathbf{r}_G(W; s)$ for test-phase windows and compare them with exported fingerprints. Shorter inference windows are desirable to reduce identification la-

tency, provided that the resulting representations remain sufficiently stable for accurate matching. In §VII, we will evaluate identification using both 8-day and 1-day inference windows.

A natural choice for comparing representations is cosine similarity, which captures proportional similarity between service usage vectors. Cosine similarity is well-suited for detecting stability during fingerprint export, where the goal is to determine whether two representations from the *same device* correspond to the same behavioral mode. However, during test-phase matching across devices, cosine similarity can be misleading when different devices exhibit similar *ratios* of service usage but differ substantially in absolute activity levels.

We observe many such cases in practice. For example, the first 8-day test window of IoT-ServTrace-2019 for Line Clova Wave yields the G-representation $\{\mathrm{TCP}/443 : 0.0421,\ \mathrm{UDP}/53 : 0.0421\}$. Using cosine similarity, this test window is most similar (score $0.9995$) to a fingerprint of the Bitfinder Awair Breathe Easy sensor that was exported based on 8-day validation windows, $\{\mathrm{TCP}/443 : 0.1667,\ \mathrm{TCP}/8883 : 0.0069,\ \mathrm{UDP}/53 : 0.1719\}$, due to the nearly identical balance between TCP/443 and UDP/53. However, Line Clova Wave has an exported fingerprint $\{\mathrm{TCP}/80 : 0.0035,\ \mathrm{TCP}/443 : 0.0469,\ \mathrm{UDP}/53 : 0.0503\}$, which is clearly closer in absolute usage to the test traffic, yet yields a lower cosine similarity because its service proportions are less perfectly balanced.

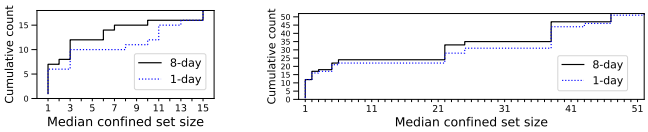
This example highlights that cosine similarity emphasizes relative proportions while ignoring magnitude differences, which can lead to systematic misclassification during test-phase matching. To address this limitation, we introduce the *Minimum-over-Maximum (MoM)* similarity. For two vectors $\mathbf{v}$ and $\mathbf{w}$, MoM is defined as:

$$\mathrm{MoM}(\mathbf{v},\mathbf{w}) := \frac{1}{|\{i|\max(v_i,w_i)>0\}|}\sum_{\{i|\max(v_i,w_i)>0\}}\frac{\min(v_i,w_i)}{\max(v_i,w_i)}.$$

MoM directly accounts for the absolute activity levels per-service and is bounded in $[0,1]$.

Applying MoM to the example above, the Line Clova Wave test window correctly matches its own fingerprint with a similarity of $0.5782$, while the similarity to the sensor fingerprint drops to $0.1658$. In our evaluation, MoM consistently improves discrimination during test-phase matching, while cosine similarity remains preferable for within-device comparisons during fingerprint export.

One may ask why the MoM similarity is not also used during the fingerprint export phase. In our experiments, applying MoM for export led to unstable behavior: to obtain any fingerprints, the similarity threshold $\theta$ had to be lowered. In addition, the re-trigger threshold had to be set below $\theta$ to prevent excessively frequent re-triggering of the exporter for some devices. However, reducing these thresholds introduces a competing effect: overly permissive similarity criteria can degrade stability in other devices, highlighting the inherent trade-off. This behavior arises because MoM is sensitive to absolute activity fluctuations, making it unsuitable for detecting stability within a device over time. In contrast, cosine similarity provides better scaling for within-device comparisons, as it captures proportional consistency while tolerating natural variations in activity magnitude. We therefore use cosine similarity for fingerprint export and MoM similarity for test-phase matching, where absolute usage levels are critical for discrimination.




(a) IoT-ServTrace-2019. (b) IoT-ServTrace-2022.

Fig. 6: Cumulative distribution of median confined set sizes for (a) IoT-ServTrace-2019 and (b) IoT-ServTrace-2022, comparing 8-day (solid) and 1-day (dotted) test windows.

## VII. Evaluating Backend-Service Fingerprints

We evaluate the proposed fingerprinting approach by progressively examining its key components. We begin by assessing how effectively service vocabularies confine the set of plausible device classes (§VII-A). We then evaluate identification using service-usage fingerprints without confinement to establish a baseline (§VII-B), and finally assess the full two-stage scheme that combines vocabulary-based confinement with fingerprint matching (§VII-C).

### A. Vocabulary-based Confined Class Labels

We first evaluate how effectively service vocabularies reduce the set of plausible device classes during identification. Given a test-phase traffic window, we apply the confinement procedure described in §V-B to restrict candidate classes whose service vocabularies are compatible with the observed services.

**Confinement effectiveness.** We begin with 8-day test windows generated using a sliding-window approach with a one-day shift. For each device, we compute the confined set size for every test window and report the median across windows. Figures 6a and 6b show the cumulative distributions of median confined set sizes for IoT-ServTrace-2019 and IoT-ServTrace-2022, respectively.

In IoT-ServTrace-2019, seven devices achieve a median confined set size of one, meaning that in at least half of their test windows, the confinement procedure reduces the candidate set to a singleton. All such devices include proprietary port numbers in their service vocabularies. In IoT-ServTrace-2022, only 12 of the 52 devices converge to a singleton median confined set, while 24 devices (approximately 46%) have median confined set sizes of six or fewer. At the other end of the spectrum, 17 devices have median confined sets equal to or larger than 26 (half of the total number of classes), indicating that service vocabularies alone provide limited discriminative power for these devices.

For IoT-ServTrace-2019, reducing the test window to 1 day yields a more polarized distribution (dotted curve in Fig. 6a). Devices tend to cluster either at small median confined set

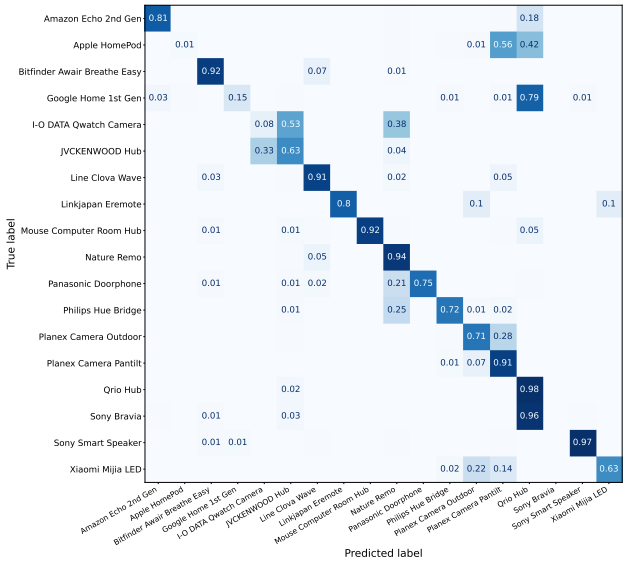



Fig. 7: Device identification is prone to significant errors when the set of plausible classes is not confined.

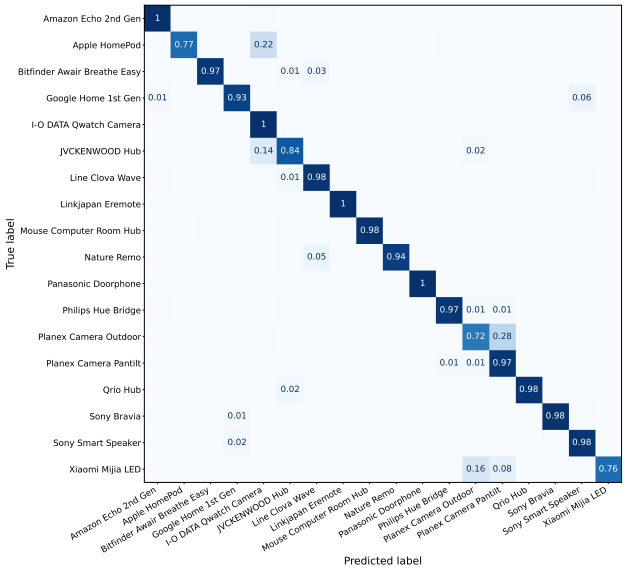



Fig. 8: Vocabulary-based class confinement improves device identification by preventing test-phase traffic from being compared against fingerprints of implausible device classes.

sizes (below four) or at large ones (above ten), with fewer devices occupying the intermediate range. For IoT-ServTrace-2022, the difference between 8-day and 1-day windows is modest, although the maximum median confined set size increases to the full class set, reflecting reduced information in shorter windows.

**Sanity of confinement.** We also assess whether the correct device label is retained within the confined set. In IoT-ServTrace-2019, three devices occasionally miss the correct label when using 8-day windows: the Planex Skamake Pan–Tilt Camera (3.85% of windows), Apple HomePod (20.88%), and the JVCKENWOOD Hub (6.59%). Using 1-day windows, the same three devices exhibit errors with rates of 1.47%, 22.40%, and 1.61%, respectively.

In IoT-ServTrace-2022, four devices show consistently high miss-out rates: Apple HomePod mini (100%), Apple TV (57.63%), Apple HomePod (55.93%), and the JVCKENWOOD Camera (48.59%) for 8-day windows, with similar rates observed for 1-day windows. Inspection reveals that these failures are caused by *concept drift* at the service level: Apple devices begin to use TCP/853 (DNS over TLS) during the test period, whereas the JVCKENWOOD Camera uses TCP/443, which was absent during training. To assess the inherent discriminative power of vocabularies under matched training/testing behavior, we exclude these devices from subsequent evaluation.

Overall, service vocabularies substantially reduce the candidate space for many devices but do not suffice for unique identification in all cases, motivating the use of service-usage fingerprints in the second stage.

### *B. Identification Using Superset of Service Usage Fingerprints*

We next evaluate device identification using service-usage fingerprints *without* vocabulary-based confinement. This setting serves as a baseline to assess whether usage fingerprints alone suffice for accurate identification, and to quantify the benefit of confinement.

As an illustrative example, Fig. 7 shows the row-normalized confusion matrix for IoT-ServTrace-2019 when 1-day inference windows are matched, using MoM similarity, against the *superset* of all exported service-usage fingerprints, which are themselves derived from 1-day validation windows. Although some device classes are identified accurately, misclassification is prevalent in many classes, indicating substantial ambiguity when confinement is omitted. In this setting, the macro-averaged precision and recall are 0.707 and 0.657, respectively.

Several errors are systematic and illustrative. Apple HomePod is frequently misclassified as the Planex Skamake Pan–Tilt Camera, despite Apple HomePod exhibiting TCP/5223 (Apple Push Notification service) in its test traffic (a service absent from the Planex camera's service vocabulary). Similarly, Google Home and Sony Bravia TV are often mislabeled as Qrio Hub. Although both Google Home and Sony Bravia TV consistently use TCP/5228 (Google Cloud Messaging), this service is not present in the Qrio Hub's service vocabulary. When vocabulary constraints are ignored, usage similarity alone is insufficient to prevent these errors.

Overall, this baseline demonstrates that service-usage fingerprints alone do not provide reliable device identification when services are shared, and usage patterns overlap. These results motivate the two-stage identification approach evaluated next, which combines vocabulary-based confinement with fingerprint matching.

### *C. Two-stage Device Identification*

**IoT-ServTrace-2019.** We now evaluate the full two-stage identification scheme, which first applies vocabulary-based confinement and then matches service-usage fingerprints of the remaining candidates. Fig. 8 shows the row-normalized confusion matrix for IoT-ServTrace-2019 using 1-day inference

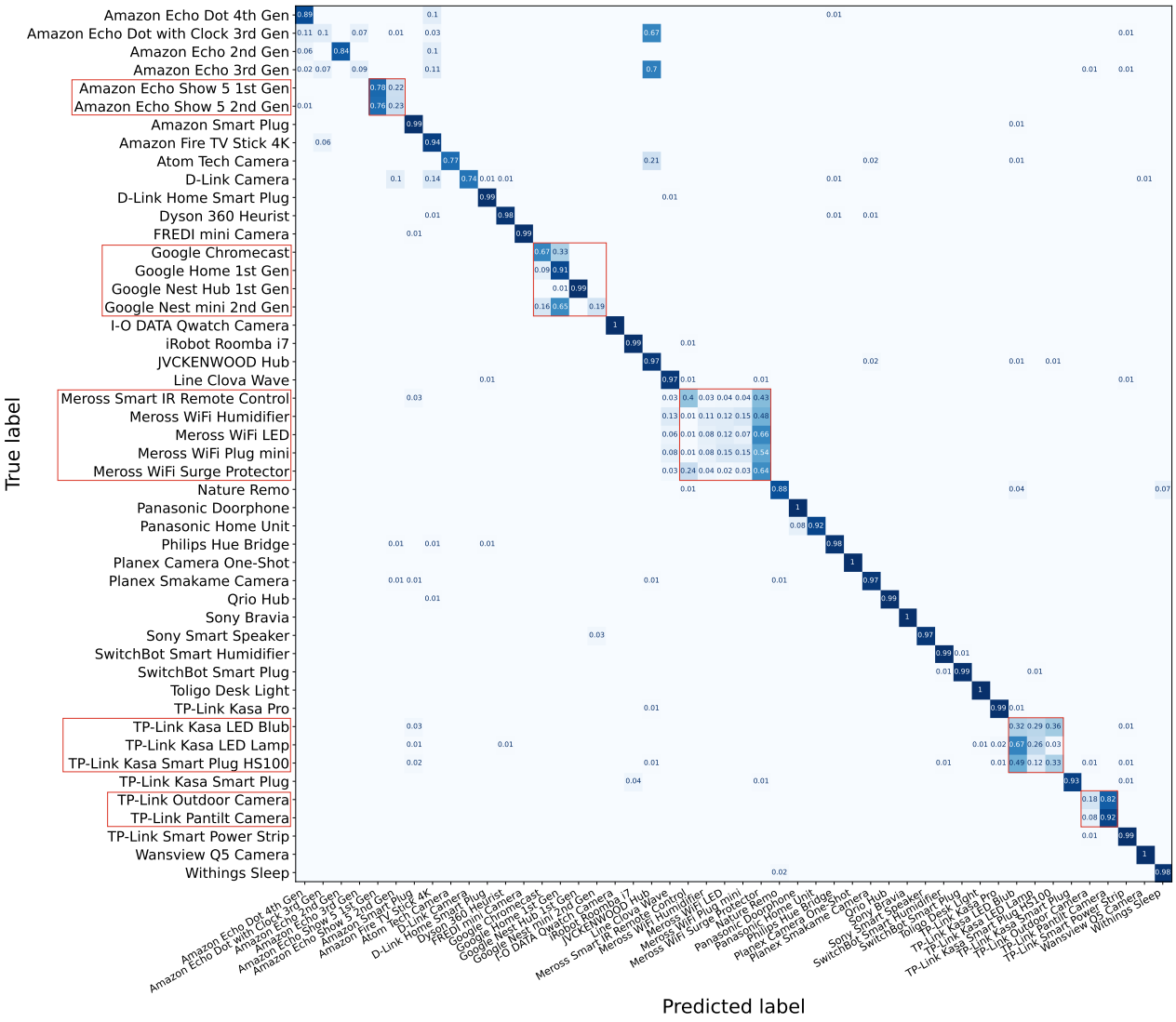



Fig. 9: Row-normalized confusion matrix for IoT-ServTrace-2022 using the two-stage device identification method with 1-day inference windows and 12-hour validation windows.

windows. Compared to the unconstrained baseline in Fig. 7, recall improves or remains unchanged for all device classes. Overall, the two-stage method increases macro-averaged precision from 0.707 to 0.937 and macro-averaged recall from 0.657 to 0.932.

Some degree of misclassification remains under the two-stage scheme. A representative example is the Planex Skamake Outdoor Camera, which is frequently misclassified as the Planex Skamake Pan–Tilt Camera. This confusion is driven by behavioral drift during the testing period: the outdoor camera exhibits occasional but recurring spikes in TCP/80 activity, causing certain inference windows to resemble fingerprints of the Pan–Tilt Camera. Interestingly, increasing the inference window size from 1 day to 8 days aggregates these spikes across more windows, reducing recall for the Outdoor Camera from 0.718 to 0.293. On the other hand, the same increase improves the recall of detecting Xiaomi LED from 0.765 (as seen in Fig. 8) to 0.964. This device relies only on TCP/80 and UDP/8053, with UDP/8053 appearing on just 32.7% of active days in 2020; consequently, many 1-day windows contain only TCP/80 traffic, leading to occasional misclassification.

**Impact of validation and inference window sizes.** The aforementioned observations illustrate that inference window size can have a non-trivial impact on classification performance. Moreover, as discussed earlier, the choice of validation window size affects the set of exported fingerprints and, thus, influences identification accuracy. Table I reports the macro-averaged F1-score for both datasets across different validation and inference window configurations. The best overall results (highlighted in bold) are obtained using 12-hour validation windows. For inference windows, the 8-day yields slightly better performance on IoT-ServTrace-2019, whereas the 1-day performs better on IoT-ServTrace-2022, which involves a significantly larger number of device classes. Importantly, for IoT-ServTrace-2019 the F1-score difference between 8-day and 1-day inference windows is only 0.005, which is negligible compared to the substantial reduction in inference latency achieved by using 1-day windows. We therefore consider the

TABLE I: Impact of validation and inference window sizes on the macro F1-score for the two datasets: IoT-ServTrace-2019/IoT-ServTrace-2022.

| 8-day validation | | 1-day validation | | 12-hour validation | | 3-hour validation | |
|---|---|---|---|---|---|---|---|
| 8-day inference | 1-day inference | 8-day inference | 1-day inference | 8-day inference | 1-day inference | 8-day inference | 1-day inference |
| 0.920/0.694 | 0.894/0.691 | 0.893/0.704 | 0.925/0.711 | **0.941**/0.718 | 0.936/**0.734** | 0.940/0.722 | 0.922/0.724 |

1-day inference window to be the preferable choice in practice. Overall, the classification performance is relatively insensitive to moderate variations in window size.

**IoT-ServTrace-2022.** We next consider the performance of the two-stage fingerprinting method on IoT-ServTrace-2022 to assess its scalability with respect to the number of classes. Fig. 9 shows the confusion matrix obtained using 12-hour validation windows and 1-day inference windows. Overall, the results demonstrate the discriminative power of backend-service fingerprints at scale. Most misclassifications occur between devices from the same manufacturer, as highlighted in the figure. In some cases, errors arise even among devices with similar functionality from the same vendor (*e.g.,* TP-Link cameras or Amazon Echo Show devices).

**Significance of multiple fingerprints.** As discussed in §VI-B, IoT devices can exhibit multiple stable service-usage modes over time, motivating the export of multiple fingerprints. This design choice proves to be important in practice. For example, in IoT-ServTrace-2019, where the best results are obtained using 12-hour validation windows and 8-day inference windows (see Table I), 85 of the 120 exported fingerprints (70.83%) are matched in correct predictions. For IoT-ServTrace-2022, using 12-hour validation windows and 1-day inference windows (which yielded the best overall performance), 114 of the 153 exported fingerprints (74.51%) contributed to correct classifications. These results show that exporting multiple fingerprints per device is both necessary and effective in capturing behavioral diversity.

## VIII. Limitations

One limitation of the proposed method is its reliance on associating traffic with individual devices using a network identifier (which is available prior to NAT) and aggregating observations over extended periods, rather than performing per-flow inference. In addition, some degree of confusion remains between devices from the same manufacturer or with similar functionality (see Fig. 9), reflecting similarities in their backend service usage. Finally, the method can be affected by temporal changes in the set of consumed services (as discussed in §VII-A).

## IX. Conclusion

We presented a backend service-based approach for fingerprinting IoT devices using passive traffic measurements. By focusing on transport-layer services that are unambiguously observable in traffic headers, our method avoids the opacity and cost of ML-based approaches and the fragility of endpoint-based techniques. A key outcome of our work is the separation between *what* services define a device and *how* those services are used over time. Each type of IoT device is characterized by a single, stable service vocabulary, while potentially exhibiting multiple service-usage fingerprints that capture distinct, locally stationary behavioral modes. This separation enables interpretable, robust, and low-latency device identification, and is validated through evaluation on 50 million IPFIX flows from 60 IoT device types collected over multiple years.

## References


[1] G. Research, "IoT Security Primer: Challenges and Emerging Practices," Tech. Rep., 2025. [Online]. Available: https://www.gartner.com/en/doc/iot-security-primer-challenges-and-emerging-practices

[2] A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE TMC*, vol. 18, pp. 1745–1759, 2019.

[3] M. Miettinen *et al.*, "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT," in *Proc. IEEE ICDCS*, Atlanta, GA, USA, Jun 2017.

[4] S. Marchal *et al.*, "AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication," *IEEE JSAC*, vol. 37, no. 6, pp. 1402–1412, 2019.

[5] V. Thangavelu *et al.*, "DEFT: A Distributed IoT Fingerprinting Technique," *IEEE IoTJ*, vol. 6, no. 1, pp. 940–952, 2019.

[6] Y. Meidan *et al.*, "A Novel Approach for Detecting Vulnerable IoT Devices Connected Behind a Home NAT," *Computers & Security*, vol. 91, p. 101968, 2020.

[7] A. Pashamokhtari *et al.*, "Inferring Connected IoT Devices from IPFIX Records in Residential ISP Networks," in *Proc. IEEE LCN*, Virtual Event, 2021.

[8] A. Hamza *et al.*, "Verifying and Monitoring IoTs Network Behavior Using MUD Profiles," *IEEE TDSC*, vol. 19, no. 1, pp. 1–18, May 2020.

[9] S. J. Saidi *et al.*, "Deep Dive into the IoT Backend Ecosystem," in *Proc. ACM IMC*, Nice, France, 2022.

[10] R. Perdisci *et al.*, "IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis," in *Proc. IEEE EuroS&P*, Virtual Event, Sep 2020.

[11] H. Guo and J. Heidemann, "IP-Based IoT Device Detection," in *Proc. ACM Workshop on IoT S&P*, Budapest, Hungary, Aug 2018.

[12] S. Landau-Feibish *et al.*, "Compact Data Structures for Network Telemetry," *ACM CSUR*, vol. 57, no. 8, pp. 191:1–31, Mar 2025.

[13] A. Nascita *et al.*, "A Survey on Explainable Artificial Intelligence for Internet Traffic Classification and Prediction, and Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 27, no. 5, pp. 3165–3198, 2025.

[14] R. Kolcun, D. A. Popescu, V. Safronov, P. Yadav, A. M. Mandalari, R. Mortier, and H. Haddadi, "Revisiting IoT Device Identification," in *Proc. IFIP TMA*, Virtual Event, Sep 2021.

[15] S. Azizi *et al.*, "Towards Label Shift Adaptation for Robust IoT Device Identification," in *Proc. ACM CPSIoTSec*, Taipei, Taiwan, Oct 2025.

[16] S. J. Saidi *et al.*, "A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild," in *Proc. IMC*, Virtual Event, USA, Oct 2020.

[17] W. He *et al.*, "Can Allowlists Capture the Variability of Home IoT Device Network Behavior?" in *Proc. IEEE EuroS&P*, Vienna, Austria, Jul 2024.

[18] A. Bremler-Barr, B. Meyuhas, and R. Shister, "One MUD to Rule Them All: IoT Location Impact," in *IEEE NOMS*, Budapest, Hungary, 2022.

[19] A. Bremler-Barr *et al.*, "It's Not Where You Are, It's Where You Are Registered: IoT Location Impact on MUD," in *Proc. ANRW*, 2023.

[20] F. Palmese, A. M. Mandalari, H. Haddadi, and A. E. Redondi, "Intelligent Detection of Non-Essential IoT Traffic on the Home Gateway," in *Proc. IEEE EuroS&P Workshops*, Venice, Italy, June 2025.

[21] S. Azizi *et al.*, "IoT Service Trace Data," https://iotanalytics.unsw.edu.au/IoTServTrace-Networking26.html, 2026.

[22] I. I-O DATA DEVICE, "Remote link files," https://play.google.com/store/apps/details?id=jp.iodata.remotelinkfiles&hl=en, 2025, google Play Store; accessed 2025-12-08.