# Combining Stochastic and Deterministic Modeling of IPFIX Records to Infer Connected IoT Devices in Residential ISP Networks

Arman Pashamokhtari, Norihiro Okui, Yutaka Miyake, Masataka Nakahara, and Hassan Habibi Gharakheili

***Abstract*—Residential Internet service providers (ISPs) today have limited device-level visibility into subscriber houses, primarily due to network address translation (NAT) technology. The continuous growth of "unmanaged" consumer Internet of Things (IoT) devices combined with the rise of work-from-home makes home networks attractive targets to sophisticated cyber attackers. Volumetric attacks sourced from a distributed set of vulnerable IoT devices can impact ISPs by deteriorating the performance of their network, or even making them liable for being a carrier of malicious traffic. This paper explains how ISPs can employ IPFIX (IP Flow Information eXport), a flow-level telemetry protocol available on their network, to infer connected IoT devices and ensure their cyber health without making changes to home networks. Our contributions are three-fold: (1) We analyze more than nine million IPFIX records of 26 IoT devices collected from a residential testbed over three months and identify 28 flow features pertinent to their network activity that characterize the network behavior of IoT devices – we release our IPFIX records as open data to the public; (2) We train a multi-class classifier on stochastic attributes of IPFIX flows to infer the presence of certain IoT device types in a home network with an average accuracy of 96%. On top of the machine learning model, we develop a *Trust* metric to track network activity of detected devices over time; and (3) Finally, we develop deterministic models of specific and shared cloud services consumed by IoTs, yielding an average accuracy of 92%. We show a combination of stochastic and deterministic models mitigates false positives in 75% of incidents at the expense of an average 7% reduction in true positives.**

***Index Terms*—IoT traffic inference, IPFIX flow records, stochastic behavioral modeling, deterministic cloud service modeling, residential networks**

## I. Introduction

HOME networks are becoming increasingly complex, yet neither ISPs nor subscribers have much visibility into connected devices and their network-level behavior. Technologies like NAT present only an opaque view of home network to the global Internet [2], [3]. This makes it surprisingly challenging for ISPs to even detect and discover connected devices, as the traffic transmitted by every active device in a home would have the same IP and MAC address of the home gateway.

A. Pashamokhtari and H. Habibi Gharakheili are with the School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, NSW 2052, Australia (e-mails: {a.pashamokhtari, h.habibi}@unsw.edu.au).

N. Okui, Y. Miyake, and M. Nakahara are with KDDI Research, Inc., Saitama, Japan (e-mails: {no-okui, miyake, ms-nakahara}@kddi-research.jp).

This submission is an extended and improved version of our paper presented at the IEEE LCN 2021 conference [1].

Consumer IoT devices have become popular by offering convenient functionalities to smart homes. It is anticipated that the number of smart homes will increase to about half a billion by 2025, which is more than two times larger than the same in 2020 [4]. Consumer IoT devices come in different categories, including but not limited to smart cameras, speakers, voice assistants, health devices, power plugs, and air-quality sensors.

IoT devices are believed to be more vulnerable [5] to cyber attacks than general-purpose information technology (IT) devices like personal computers and smartphones. This is mainly due to a lack of sufficient security measures embedded into resource-constrained IoT devices. Moreover, typical home users often do not have adequate skills to protect their network and devices, hence creating risks for the entire Internet ecosystem. Recent reports [5], [6] highlight how a significant portion of deployed IoT devices, exposed with default passwords [7], are low-hanging fruits for attackers.

Traditional static tools, used for identifying IT assets, fall short when employed for agentless IoT devices [5] – not fully capture the heterogeneous behavior of IoT assets. However, obtaining continuous visibility into connected networks is an essential first step for securing these vulnerable devices [8]. Visibility means profiling the expected behavior of IoT devices by analyzing their network activity and then using it to infer device types. Once device types are determined (classification), their dynamic behavior can be tracked (monitoring), allowing for verification of their health or flagging any significant deviation from the norm.

IoT devices in homes are able to connect (via the ISP network) with their intended servers on the Internet. Stable and fast Internet connections can indulge botnets and malware in launching coordinated volumetric attacks using millions of infected IoTs [9]. This can affect the performance of ISPs' core network and potentially make them accountable for carrying traffic of illegal campaigns [10]. On the other side, residential subscribers gradually express willingness in paying to ISPs [11] for improved security of their Internet-connected assets. Therefore, ISPs seem to be relatively incentivized to play a role in providing network monitoring and security services to households [12].

One may argue that IoT manufacturers and/or cloud providers can also offer these services. We note that the maturity of IoT manufacturers and/or cloud providers varies widely, so the entire consumer IoT market is particularly vulnerable to low-end, low-quality manufacturing and high-risk data handling practices [13]. Additionally, unlike ISPs,

IoT manufacturers and cloud providers often do not establish strong and long-term business relationships with end-users. Lastly, manufacturers and cloud providers can, at best, obtain visibility into software components embedded in their devices. However, ISPs are best poised to monitor and analyze the network behavior of connected devices and hence become capable of detecting deviations from the norm. Therefore, from both business and technical viewpoints, ISPs are a better option than IoT manufacturers and cloud providers. For users, engaging with one entity (their ISP) becomes easier than with multiple entities (manufacturers/cloud providers) to receive security services.

User privacy is an important factor for ISPs. Firstly, we envision the inference of connected devices in residential networks to be offered as a subscription service [14] to manage privacy concerns arising from the amount of personal information collected by ISPs. Users will provide their consent (and they may opt out in the future) to activate this service by accepting the terms and conditions provided by their ISP. Moreover, the ISP may choose to narrow its focus on a limited number of well-known vulnerable devices (*e.g.,* obtained from public resources like CVE or NVD). Therefore, detecting only high-risk IoT devices would benefit both users and the ISP. Secondly, the method developed by this paper (in contrast to some of the prior works [15]–[18]) only utilizes flow-based metadata (*e.g.,* flow volume and protocol) without requiring packet payloads that contain more sensitive and private information.

Given their specific functions, IoT devices (as opposed to IT devices) often display a distinct set of identifiable patterns on the network [8], enabling operators to tightly model their expected (normal) behavior [19].

Network behaviors of connected IoT devices highly depend on their expected functionality (*e.g.,* security camera, power switch, or motion sensor) and the implementation of functionalities by their manufacturer and/or developer. Therefore, one can describe device behaviors by two categories of features: (a) **stochastic** features such as distribution of packet count/inter-arrival time or total flow volume/duration; and (b) **deterministic** features such as specific services or domain names that a device communicates with them. Stochastic features display patterns in their distribution, making them conducive to machine learning techniques for modeling purposes. Hence, they are widely used by both industry and academia for enhancing network visibility in a variety of domains and use-cases [20]. On the other hand, deterministic features [8], [21]–[23] can be distinctly predictive and may not necessarily need a machine-learning algorithm to generate a model for inferring connected devices. In fact, machine learning algorithms cannot be directly applied to some deterministic features like domain names as there is no finite superset for them.

There exist prior works [3], [15]–[18], [22] on systems and methods that ISPs can employ to detect IoT devices in home networks. We will thoroughly discuss in §II their limitations particularity in terms of practicality at scale. This paper employs IPFIX data to detect consumer IoT devices in households. IPFIX is a well-known protocol for collecting flow-based metadata. IPFIX has been used by industry for many years, and hence employing its features at the edge of ISP networks requires no change to home networks. More importantly, IPFIX records do not have any payload or other sensitive (*e.g.,* user-specific) information, so that privacy concerns are minimized.

TABLE I
SUMMARY OF CLOSELY RELEVANT PRIOR WORK.

| Work | Deployment | Inference | Input data | Features |
|---|---|---|---|---|
| [3] | post-NAT | ML | NetFlow | activity + identity |
| [22] | post-NAT | DT | NetFlow/IPFIX | domain names |
| [15] | pre-NAT | DT | packets | domain names |
| [16] | pre-NAT | ML | packets | activity + identity |
| [17] | pre-NAT | ML | packets | activity |
| [18] | pre-NAT | ML | packets | activity |
| this work | post-NAT | ML + DT | IPFIX | activity + identity |

We make the following contributions: **(1)** We analyze (§III) near three million IPFIX records of 26 IoT devices in our testbed. Then, we extract 28 flow-level stochastic features from the IPFIX records. We publicly release our dataset [24], consisting of more than nine million IPFIX records; **(2)** We use these features (§IV) to train and tune a multi-class classifier model using a decision tree-based machine learning algorithm to infer the type of IoT devices from IPFIX records. We also deduce thresholds specific to each device class for reducing the misclassification rate and develop a trust metric per class, enabling us to monitor the behavioral health of detected devices; **(3)** Finally, we develop two types of deterministic models (§V), each with differentiated capabilities in predicting connected IoT devices by analyzing the cloud service information (embedded in outgoing IPFIX records) that IoT devices consume. We evaluate the efficacy of our deterministic models independently. We also show how deterministic models can mitigate false positives when combined with our stochastic (machine learning-based) model.

## II. Related Work

Gaining visibility into residential networks can enable value-add service offerings like quota management, parental control, and cyber security monitoring [2], [14], benefiting both ISPs and subscribers.

**Inferring from Residential IoT Traffic**: Consumer IoTs are purpose-built devices to perform a finite (and often distinguished) set of functions on the network, and therefore create an opportunity for ISPs to automatically profile their network behavior.

Recently, several attempts have been made by researchers [3], [15]–[18], [22] to help ISPs detect IoT devices in home networks which are summarized in Table I. The third column (Inference) indicates whether they use a deterministic model (DT) or a machine learning model (ML). The last column (Features) shows traffic attributes they employ: activity features are like packet/flow size, inter-arrival time, and flow count, whereas identity features are binary features like IP protocol (`TCP` or `UDP`) and application protocol (`HTTP`, `DNS`, and `NTP`).

Existing works come with their own limitations: (a) all (except for [22]) require hardware/software changes to individual home gateways, making them impractical for deployment at scale; (b) they [15]–[18] heavily depend on the identity

(MAC/IP address) of devices, making them not applicable for post-NAT deployment; and, (c) they [15], [22] purely rely on deterministic signatures like domain names and/or IP blocks associated with manufacturers, which may not necessarily capture the behavior of individual devices and hence yield less reliable inference.

Work in [3] requires installing dedicated hardware called a "local detector" with NetFlow enabled to identify vulnerable IoT devices in home networks using public databases like CVE and NVD. This work uses a combination of activity (like packet/byte count) and identity (like port numbers) features. However, it mixes all these features into a single ML model, which has been shown to have a detrimental effect on the prediction [8]. The same work shows how models separately trained on activity and (non-binary) identity features can improve the overall quality of prediction. In our work, we apply two different methods, namely, a stochastic model (ML) to activity features and a deterministic model (DT) to non-binary identity features, and combine their outputs to obtain the final inference.

Work in [22] develops a scalable deterministic model to infer connected IoT devices in home networks by analyzing domain names they contact. Similar to our work, the authors collect flow records (IPFIX and NetFlow) but from the core of ISP networks. Since IPFIX/NetFlow records do not contain domain names, it is necessary to perform a reverse mapping from IP addresses to domain names using tools like DNSDB and Censys, which incur additional processing costs. Another missing element of this work is that it ignores all available activity features of the flow records and solely relies on domain names that are not readily available in the records. In our work, we use both activity and identity features that are directly retrieved from the records without additional processing.

Work in [22] develops a deterministic model to infer connected IoT devices in home networks by analyzing domain names they contact. Similar to us, the authors collect flow records (IPFIX and NetFlow) but from the core of ISP networks. Since IPFIX/NetFlow records do not contain domain names, it is needed to perform a reverse mapping from IP addresses to domain names using tools like DNSDB and Censys which incur additional processing cost. Another missing element of this work is that it ignores all available activity features of the flow records and solely rely on domain names which are not readily available in the records. In our work we use both activity and identity features that are directly retrieved from the records without additional processing.

Works in [15]–[18] need to be deployed pre-NAT as they need device identity (lP/MAC address) to determine the type of individual devices. Pre-NAT deployment is not applicable at scale since it requires software/hardware change on each home gateway. Also, they analyze packets that incur inspection and processing costs and may raise privacy concerns. Work in [15] uses domain names to infer IoT devices. It needs to capture DNS response packets to create a mapping of IP addresses and domain names. IoT-Sentinel [16] classifies IoT devices by analyzing packet-based features collected by SDN-enabled gateways.

DEFT [17] developed a hierarchically distributed method for classifying IoT devices in home networks. The authors proposed to use an SDN-based home gateway in individual homes, coordinated by a central controller in the ISP cloud. Each home gateway extracts a mix of (packet and flow) features and applies a trained classifier (running as a virtual network function on the gateway) to these traffic features – when unsure (or a new device is discovered), locally computed features are sent to the central controller. AuDI [18] inspects packets to identify periodic flow (autonomous flows like `DNS` and `NTP`) and extracts several activity features over a fixed period for classifying IoT devices.

Work in [25] developed a technique for detecting home devices even if their traffic is padded and shaped (sent with a fixed transmission rate) to avoid detection. Their method can only detect the count of active devices and identify a subset of them when deployed post-NAT. To detect the number of active devices from an aggregated distribution, the inference model needs to be trained with traffic trace of all possible combination of devices which is extremely expensive in terms of computation.

Authors in [26] developed a post-NAT inferencing technique based on repetitive features occurring in packet bursts. They use packet-based features like packet size, time-to-live, `TCP` flags, and `TCP` window size. Packets within one second time window are grouped into a burst. They extract the longest common features from every two bursts and use them for inference purposes. This method requires shallow packet inspection and packet-to-packet comparison; hence, seems computationally more expensive than our flow-based approach.

Work in [27] used federated learning for training distributed models for detecting faults in industrial IoT devices. A similar approach can be applied to home networks where a local learner on each home gateway collects network traffic of devices in the network and trains a local model. Similar to [3], [15]–[18], federated learning approaches require changing home gateways which are not necessarily desirable by ISPs at scale.

**Inferring from IPFIX Telemetry**: IPFIX has been used for different purposes like flow-based anomaly detection [28] and application classification [29].

As shown in Table I, papers in [3], [22] are the closest in the literature to ours in terms of the system architecture (analyzing post-NAT flow data for classifying IoT devices in home networks). However, they only rely on a deterministic or a stochastic model without leveraging their combined capabilities. In §IV-D and §V-C, we will develop and evaluate the performance of stochastic and deterministic models specific to our IPFIX data records. These two separate models will serve as baselines (and proxies of prior work) in our experimental evaluations in §V-D, where we show how the final inference can benefit from combining the two models, achieving a $75\%$ reduction in false positives.

## III. Inferencing Connected IoT Devices from IPFIX Records

Identifying the composition of IoT devices in households can be done either by collecting network telemetry data

(packet-based and/or flow-based) from inside (pre-NAT) or outside home networks (post-NAT). The pre-NAT telemetry, indeed, reveals more information about the connected devices, particularly their unique identifiers (*e.g.,* MAC and/or IP addresses), enabling network operators to combine various telemetry records and map them to their unique device identifier. However, collecting pre-NAT telemetry is a nontrivial exercise without making changes to legacy home gateways – prohibitively expensive for ISPs to deploy at scale for millions of households. On the other hand, the post-NAT approach provides a limited amount of data, primarily because NAT hides the entire internal network and identity of active devices – this makes it infeasible to directly associate flow records with their respective end-devices. That said, post-NAT inferencing and monitoring would be practically more attractive for ISPs, particularly for the ease of deployment at scale.

IPFIX [30] is a standard protocol for transmitting flow-level metadata from network switches and routers for network monitoring and traffic analysis. IPFIX uses a 5-tuple key (source and destination IP address, source and destination transport-layer port number, and transport-layer protocol) to uniquely identify and aggregate packets belonging to the same flow. For connection-oriented flows, IPFIX uses session termination signals such as `TCP FIN` to detect the end of flows. For connection-less flows (*e.g.,* `UDP`), on the other hand, IPFIX uses two parameters, namely *idle-timeout* and *active-timeout*. If no packet is exchanged over a flow for the idle-timeout or if a flow is active for more than the active-timeout, the flow is terminated and the corresponding IPFIX record is exported.

We choose IPFIX telemetry to infer connected IoT devices in home networks for the following reasons: (i) the industry has increasingly adopted IPFIX as a multi-vendor universal lightweight protocol for performance and security monitoring, and hence it is perceived less challenging for ISPs to enable it on their network; (ii) IPFIX telemetry can be employed at (the edge of) ISP networks without the need to make any changes to subscriber home networks (gateways), and hence scalable; and (iii) IPFIX records only contain metadata of network communications without revealing any payload or other sensitive information, and hence relatively low risks to user privacy. Nevertheless, relying on IPFIX comes with its own challenges: (a) collecting IPFIX telemetry from the (edge of) ISP networks will provide partial visibility into the activity of home networks as the identity (MAC/IP) of connected devices will be hidden by NAT routers, and (b) IPFIX records provide aggregate information about flows, yielding coarse-grained data compared to fine-grained packet traces.

Fig. 1 illustrates the architecture of our inference system. Internet traffic of home networks is exchanged with the ISP network, where IPFIX-enabled edge routers are configured to export the respective IPFIX records. The inference can be made from a single IPFIX record (identifying the corresponding device that generated the flow) or from a collection of IPFIX records during a fixed epoch time (say, daily). Details will be discussed in §IV and §V where we explain our stochastic and deterministic models. The ISP will progressively construct a set of IoT devices for each home upon discovering a new device type. It can be seen in Fig. 1 that how post-NAT

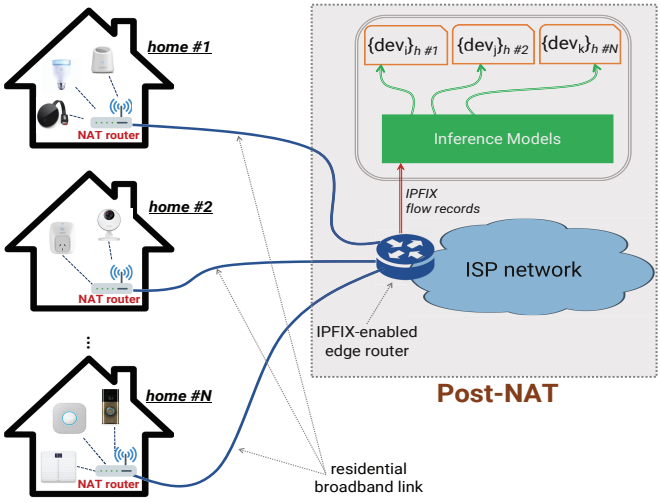



Fig. 1. System architecture of post-NAT inference from IPFIX records.

inferencing (the scope of this paper) can be employed by the ISP, capturing only remote flows via which an IoT device inside a home network communicates with a remote server on the Internet. We note a pre-NAT approach (practically challenging for ISPs, and beyond the scope of this paper), in contrast, may provide richer visibility into all communications, including both local and remote flows. Obtaining visibility into local traffic of home networks, however, can raise privacy concerns for users.

In practice, traffic inference models (particularly for determining the type of connected IoT devices) may encounter challenges in response to changes in (a) the composition of IoT devices (users may add or remove one or more devices) or (b) the network behavior of an existing device can evolve over time due to firmware upgrades or configuration updates.

Let us start with the first scenario of change. The inference task is unaffected if a known type of device is added to or removed from the network. However, connecting an entirely new type (unknown to the model) will go undetected. We note that traditional supervised learning algorithms (this paper included) aim to train a model in the closed-set world, where training and test samples share the same label space. Open set learning is a more challenging and realistic problem, where there exist test samples from the classes that are unseen during training. Automatic recognition of open set [31] is the sub-task of detecting test samples that do not come from the training, which is beyond the scope of this paper. Therefore, in this paper, it is not unexpected if some devices in a given home network remain undetected (false negative), but it is important for the ISP that their model does not detect nonexistent devices (false positive). This is crucial since inferring the composition of devices in a home network may lead to subsequent actions like notifying users about a vulnerable device on their network. Hence, reducing false positives is one of the primary objectives of this paper.

Moving to the second scenario which we assume it to be relatively easier to detect by a Trust metric (will be discussed in (§IV-C and §IV-D). Again, re-training the model by incorporating behavioral changes [32] and/or variations [23] is beyond the scope of this paper.

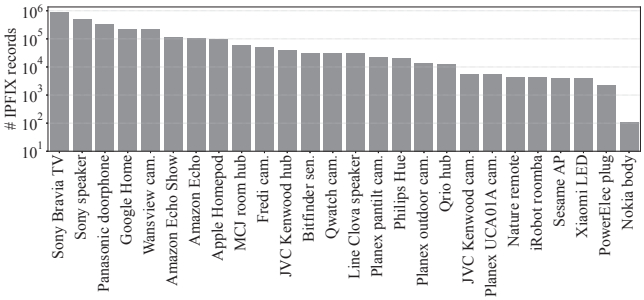



Fig. 2. Distribution of remote IPFIX records across 26 IoT devices.

### A. IPFIX Dataset and Features

Our residential testbed in Japan[1] comprises 29 consumer IoT devices ranging from smart cameras, speakers, door phones, and TV to lightbulbs, sensors, and vacuum cleaners. We collected PCAP (Packet Capture) traces from our testbed (pre-NAT) during January, March, and April 2020. Due to technical issues in our lab, we could not collect the network traffic during February 2020. It is important to note that we collected our data in a pre-NAT setup to obtain ground-truth label (device MAC/IP address) of traffic data; however, we do not take any advantage of those identities in our inference process. We exclude the local flows from our raw data collected pre-NAT, and only keep remote flows for developing our inference models in this paper. In other words, our processed data truly represent a post-NAT telemetry scenario.

Our IoT network traces constitute: (i) traffic generated due to human users interacting with the devices at least twice a week in January and March – for example, asking questions from smart speakers, checking air quality from smart sensors, switching the color of smart lightbulbs, and watching a live stream from cameras; as well as (ii) traffic generated by the devices autonomously – for example, `DNS/NTP` activities that are unaffected by human interactions. During April, due to the COVID outbreak, devices were operated completely autonomously. For April 12-13 and 21-22, we have no traffic trace of the devices due to a power shutdown and the server outage, respectively. Note that PCAP files were configured to record up to a size limit of 9.6 MB (meaning time slots with a duration of 20-30 minutes). Hence extracting IPFIX records from individual PCAPs may lead to some flows that cross the boundary of subsequent slots and break into shorter incomplete flows. We, therefore, stitch PCAPs on a monthly basis, and then extract IPFIX records from the monthly PCAPs. We use YAF (Yet Another Flowmeter)[2] tool [33] to generate IPFIX binary files from the PCAP files, followed by applying Super Mediator tool [34] to generate IPFIX JSON (JavaScript Object Notation) files from the binaries. YAF comes with an option called `flow-stats` that allows exporting a richer set of flow-level information. We use this option to embed all the possible features into individual IPFIX records. YAF tool defines default values of idle-timeout and active-timeout parameters (for `UDP` flows) equal to 5 and 30 minutes, respectively.

Our IPFIX dataset [24] contains more than nine million IPFIX records in which about $70\%$ of them are local flows that do not leave the NAT gateway (cannot be captured by the ISP edge routers). We use IPv4 address element of IPFIX records to filter local flows, including: (i) unicast flows with both source and destination from private address space (*i.e.,* `10/8`, `172.16/12`, `192.168/16`) reserved by the Internet Assigned Numbers Authority (IANA) [35], (ii) multicast flows (*i.e.,* with address in the range of `224/4`), (iii) link-local flows (*i.e.,* with address in the range of `169.254/16`), and (iv) broadcast flows. Three devices, namely Sony camera, Planex one-shot camera, and LinkJapan eSensor are found with only local flows in the dataset, hence excluded from our study in this paper. After excluding the local flows, there are about three million remote IPFIX records. For the rest of this paper, we will only focus on remote flows (nearly three million IPFIX records). Note that IPFIX records are bi-directional, but we can determine the initiator of each communication flow by checking the source IP address. In our dataset, $94\%$ of the flows are initiated by IoT devices (inside the home network), and the remaining $6\%$ (traversing NAT) are initiated by remote cloud servers. We will explain in §V how NAT traversing flows add noise to inputs of our deterministic model.

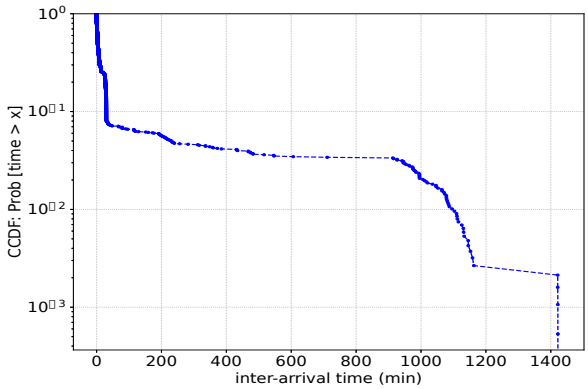



Fig. 3. CCDF of (daily) average inter-arrival time of IPFIX records per IoT device in our dataset.

Fig. 2 illustrates the number of IPFIX records (remote flows) per device collected during January, March, and April 2020. Highly active devices like Sony Bravia TV, Sony speaker, Panasonic doorphone, Google Home, and Wansview camera generated over 100K flows. Devices like MCJ room hub, Fredi camera, JVC Kenwood hub, Philips Hue, and Qrio hub are moderately active with 10K-100K flows. Lastly, we observe relatively less-active devices like Nokia body that generated only 111 flows during these three months.

Considering the frequency of network activities per device type, we plot in Fig. 3, CCDF (Complementary Cumulative Distribution Function) of average flow inter-arrival time, on a daily basis per device during the three months. The average inter-arrival time across all device types is about 50 minutes, while 14 device types generate a flow every 10 minutes on average. We observe that the longest daily average of flow inter-arrival time belongs to Nokia body with more than 14 hours because this device only becomes active whenever the user interacts with it.

Different IoT devices display distinct behaviors on the network. Researchers have shown how network activity (both in terms of total traffic volume and temporal patterns) of

[1]Cybersecurity lab of KDDI Research in Saitama, Japan.

[2]YAF was created as a reference implementation of the IPFIX protocol, providing a platform for experimentation and rapid deployment of new flow meter capabilities.

TABLE II
"UNIDIRECTIONAL" ACTIVITY FEATURES EXTRACTED FROM IPFIX RECORDS.

| Feature | Description |
|---|---|
| `packetTotalCount` | # packets |
| `octetTotalCount` | # bytes |
| `smallPacketCount` | # packets containing less than 60 bytes payload |
| `largePacketCount` | # packets containing at least 220 bytes payload |
| `nonEmptyPacketCount` | # packets containing non-empty payload |
| `dataByteCount` | total size of payload |
| `averageInterarrivalTime` | average time (ms) between packets |
| `firstNonEmptyPacketSize` | payload size of the first non-empty packet |
| `maxPacketSize` | the largest payload size |
| `standardDeviationPayloadLength` | standard-deviation of payload size for up to the first 10 non-empty packets |
| `standardDeviationInterarrivalTime` | standard-deviation of inter-arrival time for up to the first 10 packets |

a smart camera or TV would differ from that of a motion sensor or air quality monitor [8], or how various manufacturers tend to embed a (relatively unique) combination of standard and propriety network services (*e.g.,* `TCP/443` for `TLS`, or `UDP/56700` for cloud control messages) in the firmware of their devices [23]. In other words, the behavior of IoT devices can be profiled using a combination of stochastic and deterministic features. Generally, stochastic features are suited for machine learning models as their patterns can be characterized by certain distributions. In contrast, deterministic features may not always be a good fit for machine learning algorithms, especially when their dimension is high. For example, in the case of network services, we require over 65K binary features (total number of possible port numbers [36]). For domain names, on the other hand, one cannot even define a bounded set of features. Deterministic features, therefore, need to be modeled differently.

In this paper, we use cloud services consumed by IoT devices for our deterministic model. Each cloud service (*e.g.,* `UDP/10086`, `TCP/8443`) is a pair of transport-layer protocol and port number on the server-side. One may choose to use other deterministic features like IP address and/or domain name of cloud servers. However, cloud IP addresses can be relatively dynamic [22], hence unreliable over time. Also, domain names are not readily available in IPFIX records, though can be retrieved by way of reactive lookups. Cloud services, instead, can be directly deduced from IPFIX records and are more likely to be persistent during the lifetime of IoT devices. In fact, network services that IoT devices offer and/or consume along with local/Internet endpoints they communicate with, constitute a formal description, called Manufacturer Usage Descriptions [37], that can be used to track IoT devices [23] or limit their attack surface [38].

TABLE III
"BINARY" IDENTITY FEATURES OF POPULAR SERVICES EXTRACTED FROM IPFIX RECORDS.

| Network service | Protocol | Transport-layer port numbers |
|---|---|---|
| `HTTP` | `TCP` | 80, 8080, 8008, or 8888 |
| `TLS` | `TCP` | 443, 1443, 8443, or 55443 |
| `DNS` | `UDP` | 53, or 5353 |
| `NTP` | `UDP` | 123 |
| others | `TCP` | *any* |
| others | `UDP` | *any* |

## IV. STOCHASTIC MODEL OF IOT NETWORK BEHAVIOR

### A. Baseline model

Our stochastic model (generated by a machine learning algorithm) characterizes the activity behavior of connected IoT devices on the network from flow-level statistical features provided by IPFIX records. We use this as a baseline and also representative of stochastic models used in prior works like [3]. Table II summarizes a list of all potential elements (indicative of flows' network activity) that will be used as a part of the features in §IV-D. It is important to note that each IPFIX record represents a bi-directional flow [30], hence there are two sets of these features – each specific to a direction (outbound and inbound). For example, `smallPacketCount` (third row in Table II) shows the number of small packets that an IoT device (the initiator of flow) transmitted to an Internet-based server (remote endpoint), and accordingly, `reverseSmallPacketCount` indicates the number of small packets that the remote endpoint sent to the IoT device on the same flow.

In addition to the activity features listed in Table II, we capture six binary identity features of each flow record (Table III), highlighting whether their network service is one of four popular services (`HTTP`, `TLS`, `DNS`, `NTP`) or *other* `TCP/UDP` services – specific network services will be analyzed by a separate deterministic model in §V. Considering popular services in our dataset, `TLS` dominates by contributing to 35% of flows, followed by `DNS` (20% of flows), `HTTP` (6% of flows), and `NTP` (2% of flows). Also, in terms of other services, `UDP` outweighs `TCP` by contributing to 30% of flows versus 3%. Note that the remaining 4% of flows use network control protocols like `ICMP` and `IGMP` that are not associated with either of the dominant transport-layer protocols namely `TCP` and `UDP`.

We found some device types display distinct behaviors by certain features in their IPFIX records. For example, IPFIX records with `reverseSmallPacketCount` larger than 500 would belong to Sony speaker with a probability of 95%; also, 97% of the IPFIX records which their `reverseDataByteCount` falls between 300 to 400KB, belong to Sony Bravia TV. That said, we observe some overlap of features across device types. This is mainly because each IPFIX record contains an "aggregate" set of information on the activity and identity of a single network traffic flow. For example, we found 16 devices share the `averageInterarrivalTime` feature, having a value less than 10 seconds on average; five devices have this feature valued between 10 and 20 seconds; and, five other devices have a value greater than 20 seconds in their IPFIX records.

In another example, we have three categories of devices in our dataset by only considering the `reversePacketTotalCount` feature: (a) Planex UCA01A camera with a high count (more than 200) of packets, (b) Xiaomi LED and iRobot Roomba with medium packet counts (between 100 and 200), and (c) other 23 device types with low packet counts (less than 70). Also, in terms of network services, devices like Fredi camera, JVC Kenwood camera, Sesame Access Point (AP), Xiaomi LED, and all three Planex cameras never use `TLS`. Instead, all IPFIX records of a device like Qrio hub are `TLS`.

These examples highlight that determining the device type from IPFIX records would certainly require a collection of stochastic features. We, therefore, employ machine learning techniques that have been widely adopted by researchers for network traffic inferencing [39] to learn the stochastic attributes of IPFIX records for inferring their device type.

We choose Random Forest, a powerful machine learning model that builds a decision tree ensemble, allowing for the automatic creation of a sequential combination of feature rules (similar to what we did in the examples above) to predict device classes. In order to maximize the prediction performance, we extract every possible feature from the IPFIX records and let the Random Forest algorithm construct the optimal decision trees.

### B. Confident Classification

As mentioned earlier, analyzing flows in a post-NAT scenario will reveal partial information about actual IoT devices connected to the home network; therefore, a machine learning-based model may misclassify some IPFIX records. Note that the primary objective of our inferencing is to determine whether an IoT device is present and active in a home network, or not. Therefore, the quality of each prediction is of top priority. This objective necessitates a method for discarding inconfident predictions, made primarily due to a lack of distinct features in the subject IPFIX flow. Note that one may attempt to aggregate individual IPFIX records over a time window in order to increase the amount of data passed into the inference model. However, this is not applicable in post-NAT deployment as there is no association between individual IPFIX records and devices.

Machine learning models often produce a measure of confidence for their prediction. The confidence-level is a number between 0 and 1. Higher confidence values indicate more reliable predictions. We, therefore, use the confidence-level of our Random Forest model to decide whether a prediction is "accepted" or "discarded" – certain thresholds are determined and applied to the confidence-level of the model's prediction. We note that thresholds can be applied globally [8] across all classes, or specifically per individual classes. Given the diversity of network behaviors as well as richness of training data across various IoT types, we choose to apply class-specific thresholds that are obtained from the training phase.

$C_L$ denotes the model's confidence for correctly predicting instances of class $L$, and the corresponding threshold is denoted by $\lambda$, which is calculated for each class separately. To set the thresholds, we use the distribution (average: $\mu_{C_L}$ and standard deviation: $\sigma_{C_L}$) of confidence per each class during the training phase as the baseline. In this paper, we consider two ways of thresholding per each class: (a) greater than the average confidence: $\lambda_1 = [\mu_{C_L}, 1]$, and (b) within $\sigma_{C_L}$ from the average confidence: $\lambda_2 = [\mu_{C_L} - \sigma_{C_L}, \mu_{C_L} + \sigma_{C_L}]$. Note that $\lambda_1$ is only concerned about its lower-bound $\mu_{C_L}$ without imposing an upper-bound. Instead, $\lambda_2$ aims to conform to the distribution obtained from training by imposing both lower and upper bounds to the model's confidence. Note that one can choose other thresholds for this purpose. In §IV-D, we will show how each of these thresholds would improve the accuracy of the model by discarding the majority of mispredictions.

### C. Behavioral Changes of IoT Devices

Our primary objective is to identify IoT devices connected to home networks using IPFIX flow records collected post-NAT. ISPs may wish to obtain additional insights into the network behavior of classified devices in each home. For example, they may want to track the activity pattern of discovered devices for purposes like ensuring their cyber health or triggering the model re-training process due to some behavioral changes. Those follow-up actions (health assurance and/or model re-training) are beyond the scope of this paper. We note that tracking the behavior of IoT devices post-NAT can be relatively challenging as our unit of data (IPFIX flow record) does not carry the identity (e.g., MAC/IP address) of their respective end-device. In the absence of a solid meta-data for associating the IPFIX records with devices, we again resort to the output of our inference model.

For reasons discussed above, a behavioral change can only be determined based on a set of observed IPFIX records per home. As our unit of data is a single IPFIX record, we need to accumulate a handful of them in order to determine a behavior change. We, therefore, choose a configurable epoch (*e.g.,* a day) to construct a collection of model predictions and look for changes in the time domain. One may choose to employ the widely-used *Jaccard index* to compare two sets of categorical values (predicted devices across successive epochs). This index quantifies the ratio of the size of the intersection of two sets to the size of their union. Let us start by formally defining the baseline Trust metric ($T_{base}$) as follows:

$$T_{base} = \frac{|P_{i-1} \cap P_i|}{|P_{i-1} \cup P_i|} \tag{1}$$

where $P_{i-1}$ and $P_i$ are the sets of devices classified for epochs $i-1$ and $i$, respectively, and $|.|$ operator returns the size of the given set.

$T_{base}$ is relatively simple and keeps track of predicted devices per household over consecutive epochs, making it a stateful measure. However, there are four caveats where $T_{base}$ falls short of expectations: (1) $T_{base}$ is computed across all devices of a home network for a given period without giving any Trust indication for individual devices; (2) if a device is removed from the network (turned off) by the user following days of being active (and hence correctly predicted),

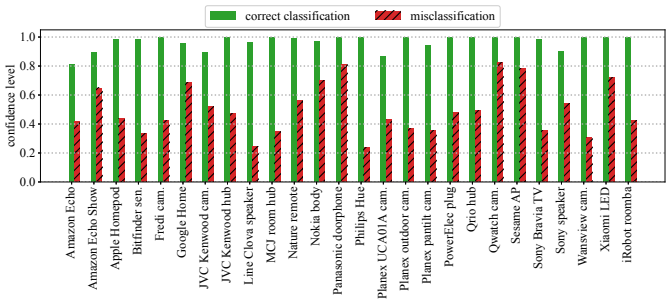



Fig. 4. Average confidence-level of correctly and incorrectly classified instances per device type for the training dataset.

$T_{base}$ will immediately drop for the next day, which is not highly desirable; (3) suppose an active device generates less (or more) records than its normal pattern (captured behavior during the training phase). In that case, $T_{base}$ will not detect such changes as long as the device is inferred by at least one IPFIX record; and (4) when the number of flows for a particular device remains consistent but flow features like the number of packets/bytes changes, $T_{base}$ would be unable to detect those dynamics.

With the shortcomings of the baseline metric discussed above, we now aim for a more comprehensive metric that focuses on capturing the state of predictions within an epoch without directly incorporating temporal dynamics. With respect to the first problem, our Trust metric needs to device-centric instead of home-centric. For the second problem, we define Trust in such a way as to be stateless, meaning it is measured for each epoch purely based on the predictions of that epoch without looking at the past. In order to address the third and fourth problems, we incorporate the number of records being predicted as the target device we aim to measure Trust for, as any severe deviation in the number of records and/or flow features will be reflected in the model's output. In §IV-D, we evaluate $T_{base}$ against a more desirable metric we will develop next and demonstrate how our metric of Trust can reveal more information about changes in the behavior of individual devices.

Considering the points highlighted above, we define Trust for individual devices over each epoch. For a device, the Trust metric is computed based on the number of predictions for that type during a given epoch (*e.g.,* a day) while taking into account class-specific patterns obtained from the training phase. For a device class $L$ in our model, we measure the expected (average) number of IPFIX records (denoted by $N_{e,L}$) and the expected rate of discarded IPFIX records (denoted by $D_{e,L}$) during training. Note that some predictions are discarded due to a lack of confidence reported by the model. A raw measure of trust is computed by:

$$T_L = \frac{N_{o,L}}{N_{e,L} \times D_{e,L}} \tag{2}$$

where, $N_{o,L}$ is the number of "observed" flows predicted as class $L$ – obviously, those predictions receive an accepted level of confidence from the model. The intuition behind this raw measure of trust is to check the number of expected flows from a device with the number of flows that are indeed classified as that device type during the epoch. Trust values around one imply that corresponding devices are behaving as expected. However, the measure of raw trust can become larger than one when a device is over-active (*e.g.,* under attack), or its flows are mispredicted as another device class. Either of these cases (*i.e.,* having the raw trust value more than one) needs to be interpreted as misbehavior. Therefore we need to normalize the raw trust so that it decays in case of deviation from normal behavior. We define normalized trust (will be referred to as trust in the rest of this paper) by:

$$T_{norm,L} = \exp\left(-\frac{(T_L - 1)^2}{2\sigma^2_{T_L}}\right) \tag{3}$$

where $\sigma_{T_L}$ is the standard-deviation of $T_L$, computed during the training phase. Use of $\sigma_{T_L}$ allows for customizing the change rate of trust across various device types – for a complex device like smart TV that displays a wide range of network activities the trust metric changes slower, while for a simple device like smart sensor it is will change relatively faster.

TABLE IV
IMPACT OF $\lambda_1$ AND $\lambda_2$ THRESHOLDS ON TRAINING INSTANCES.

| | **Accepted** | | **Discarded** | |
|---|---|---|---|---|
| | **Correct** | **Incorrect** | **Correct** | **Incorrect** |
| $\lambda_1$ | 1,325,787 | 924 | 237,724 | 100,559 |
| $\lambda_2$ | 1,408,600 | 28,214 | 154,911 | 73,269 |

### D. Evaluating Efficacy of Stochastic Modeling

For performance evaluation, we split our data into two groups: training (1,664,994 IPFIX flows recorded in January and first half of March), and testing (1,416,479 IPFIX recorded in the second half of March and full April). We use Scikit-Learn Python library to train our Random Forest classifier.

*1) Stochastic Model Training and Tuning:* We tune the parameters of our Random Forest model to maximize its performance for the chosen features (Table II and III). We tune three important parameters, namely the number of decision trees, the maximum number of attributes to consider at each branch split, and also the maximum depth of each decision tree. For each combination of parameters, we quantify the model accuracy by 10-fold cross-validation, whereby the training dataset is randomly split into training (90% of total instances) and validation (the remaining 10% of total instances) sets, and the accuracy is averaged over 10 runs to produce a single performance metric. We compute the model's accuracy by averaging the rate of correct predictions (true-positives) across individual classes of our model. In other words, the average of all values across the main diagonal of the model's confusion matrix, shown in Fig. 5. We found the best tuning parameters which yield the highest prediction accuracy (88%) to be 100 decision trees, maximum of five features for finding the best split, and maximum depth of 20 for each decision tree.

We next quantify the confidence of the model (on the training dataset) for correct classifications as well as incorrect classifications across various classes of device type. As shown in Fig. 4, green bars indicate the average confidence for

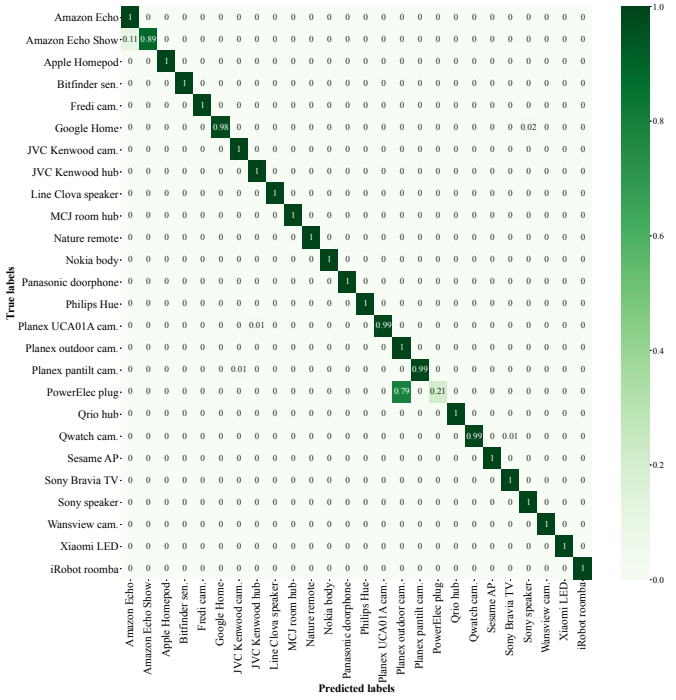



Fig. 5. Confusion matrix of accepted predictions – Random Forest model followed by applying class-specific confidence thresholds ($\lambda_1$).

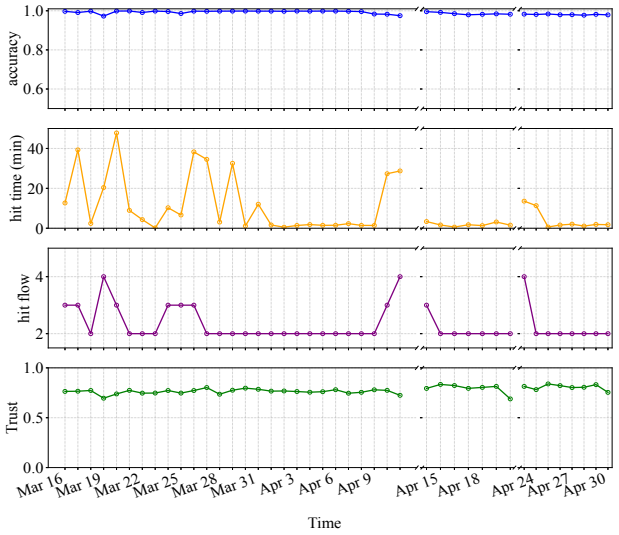



Fig. 6. Time-trace of average accuracy, trust, hit-time, and hit-flow of our inferencing across all classes on the testing dataset.

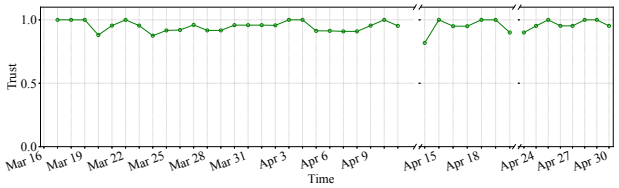



Fig. 7. Time-trace of baseline trust ($T_{base}$) during the testing period.

correctly predicted flows and red bars indicate the average confidence for mispredicted flows. A clear takeaway from this analysis is that the model is more confident when a flow is correctly classified and is relatively less confident when a flow is misclassified.

The second observation is that the two threshold methods ($\lambda_1$ and $\lambda_2$ we defined in §IV-B) affect our inferencing differently, as shown in Table IV: $\lambda_1$ is found to be more conservative by discarding $20\%$ of predictions, compared to $14\%$ discard rate resulted from applying $\lambda_2$; $\lambda_1$ is proven to discard more of incorrect predictions compared to $\lambda_2$ ($99\%$ versus $72\%$); considering the accepted predictions, $\lambda_1$ gives a slightly better accuracy than $\lambda_2$ ($99.91\%$ compared to $99.02\%$). Given the primary objective being the quality (purity) of predictions, we use the first method of thresholding ($\lambda_1$) for the testing phase of our evaluation.

*2) Stochastic Model Testing:* Having tuned the parameters and obtained the class-specific confidence thresholds, we now evaluate the efficacy of the stochastic model by applying it to our testing dataset. Fig. 5 shows the confusion matrix of our inferencing (the model combined with $\lambda_1$ thresholds). The average accuracy across all classes is $96\%$, with 20 classes receiving more than $99\%$ accuracy – the rate of correct predictions across accepted predictions. We found that applying the thresholds significantly improves the quality of model prediction. Note that the average accuracy of the model alone (without $\lambda_1$ thresholding) is $82\%$. Also, only $1\%$ of the accepted instances are incorrectly predicted. However this quality of prediction is achieved at the cost of discarding a part of the correctly predicted flows *i.e.,* $26\%$ of correct predictions when the model is less confident than expected thresholds. Discard rate of correct predictions, on average, is about $17\%$ per class.

Though our stochastic model gives an overall acceptable performance, we find flow records of the Power Elec plug are not well predicted. We found that the model mispredicts most of Power Elec plug instances (relatively confidently) as Planex outdoor camera. Further investigations revealed that almost all of those misclassified flows correspond to `UDP/10001` service which is the second top service in IPFIX records of Planex outdoor camera (the mispredicted class). Analyzing the features of `UDP/10001` records from these two classes showed that the inter-arrival time and the number of outgoing packets in the flows of Power Elec plug significantly changed from the training to the testing phase, becoming similar to those of Planex outdoor camera flows, and hence resulted in the misprediction. Note that the ISP may choose to re-train the model [32] after verifying certain legitimate changes, but re-training is beyond the scope of this paper.

Another takeaway from our evaluation is that the performance of machine learning-based inference model varies across different network services. The scheme yields a better accuracy in `TCP` flows compared to that of `UDP` flows. Focusing on `TCP`, we see the highest true-positive rate in both `TLS` and "other" `TCP` flows with $99\%$, followed by `HTTP` flows with $94\%$. For `UDP` flows, "other" `UDP` services have $100\%$, followed by `DNS` and `NTP` with $98\%$ and $72\%$ true-positives, respectively. It is important to note that `NTP` flows receive the highest discard rate ($99\%$ of the `NTP` flows are predicted with fairly low confidence). This is mainly because the vast majority of the `NTP` flows have the exact size of 76 bytes (including payload, `UDP` and `IP` headers) across all IoT classes in our dataset. We note that most of these IoT devices use the

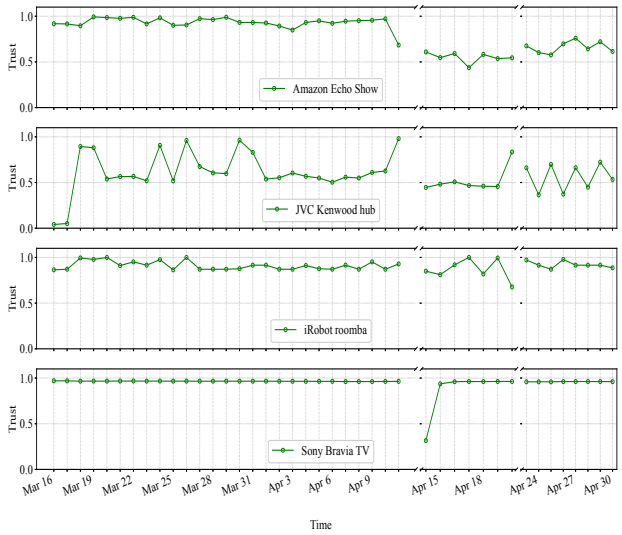



Fig. 8. Daily measure of Trust for four representative IoT devices on the testing dataset.

minimum packet structure defined by the `NTP` standard without any extension field, resulting in identical byte count feature for this specific type of flows.

*3) Operational Insights:* We track the efficacy of our inference model in operation on a daily basis, as shown in Fig. 6. In addition to overall accuracy (top plot), which remains fairly high during our testing period, we capture two metrics: (a) *hit-time*, the time taken for our model to detect a device since its first-seen on the network, and (b) *hit-flow*, which indicates the number of flows generated by a device before the model correctly detects it. It is seen from the second top plot in Fig. 6 that the average hit-time across all classes varies between 9 seconds and 48 minutes, highlighting the responsiveness of our inference scheme. Also, we observe that the average hit-flow mostly equals 2 (with a maximum of 4) – this suggests that discarding about a third of correctly classified flows does not incur an excessive delay in our inference.

Moving to the trust metric shown in the bottom plot of Fig. 6, we see consistently high values during the testing period on average (highlighting the overall health of devices). Only on 20th April, we observe a slight drop in the trust due to the early shutdown of our capturing server in preparation for a planned power outage on 21st and 22nd April – this resulted in just 3 hours worth of data captured on 20th April.

Fig. 7 shows the temporal dynamics of baseline Trust ($T_{base}$). Based on its definition, $T_{base}$ starts getting value from the second day of the testing period (17th March). It can be seen that $T_{base}$ displays relatively high and consistent values across the entire testing period, with a daily average of 95%, indicating most devices are consistently present and active in the network. In what follows, we will see certain changes in the behavior of some devices that will go undetected by $T_{base}$ (mainly because of its aggregate nature). In contrast, our metric $T_{norm,L}$ (by incorporating fine-grained contexts) is able to highlight them.

**Dynamics of Trust Metric:** To better understand the temporal dynamics of the Trust metric, we plot in Fig. 8 the daily Trust for a few representative classes which display noticeable changes in their behavior.

We categorize the behavioral changes by trust metric in two groups: (i) permanent declines which are probably caused by a legitimate firmware upgrade, leading to change in traffic features and/or the number of generated flows, and (ii) temporary declines that can be caused by an intermittent network condition or device usage/configuration for a short period of time, leading to generation of significantly different number of flows without necessarily affecting the traffic features. Fig. 8 illustrates the daily trust of four representative devices from our testbed: Amazon Echo Show (change in traffic features), and JVC Kenwood hub (change in the number of flows) represent a firmware upgrade scenario, while iRobot roomba and Sony Bravia TV represent temporary behavioral/usage changes affecting their flow count.

Amazon Echo Show has an average trust of 0.94 before April 11 on which it starts to decay and remains around 0.61 until the end of April. By further analysis, we found that the daily flow count was almost expected, but some significant changes were observed in certain traffic features (`reverseOctetTotalCount` and `reverseDataByteCount` dropped by about 80%) compared to the training phase. These changes caused the number of accepted flows to drop by 40%, compared to the expected value from the training phase.

JVC Kenwood hub starts with a very low trust value on 16th and 17th March, followed by a fluctuating trust measure averaged at 0.66 from 18th March to 11th April. Next, we observe a permanent drop, starting from 14th April with an average trust of 0.54. JVC Kenwood hub generated 983 and 1019 flows during the first two days respectively (16th and 17th March) – in each day, about 870 flows received accepted prediction. Moving to the second half of April, the hub generated on average 218 flows per day, and around 168 of flows (daily) are predicted with an acceptable confidence. These values are far from the expected behavior of JVC Kenwood hub based on the training dataset where the expected number of flows is about 415 per day.

The iRobot Roomba displays a consistently high trust value over the testing period (about 0.91 on average) except for 20th April (with trust value of 0.67). This device has an average of 50 flows per day over the testing period, except on 20th April, it only had 7 flows. The shutdown of our capture server in preparation for a planned power outage on the following days caused this drop. As the expected flow count for iRobot Roomba is about 43 per day, the significant deviation on 20th April resulted in a low trust value for this device.

Sony Bravia TV shows a persistent trust measure throughout the testing phase except for April 14, when it generated a very high number of flows. On average, it generated 5,648 flows per day during those days with a healthy trust metric. However, the flow count rose to 98,013 on April 14 – of those flows, ≈44,000 (about 3.8 times the expected baseline) received an accepted prediction.

## V. Deterministic Model of IoT Cloud Services

In the previous section, we trained a multi-class ML model on stochastic features of IPFIX records. Although ML models are popular in the literature, they come with some open

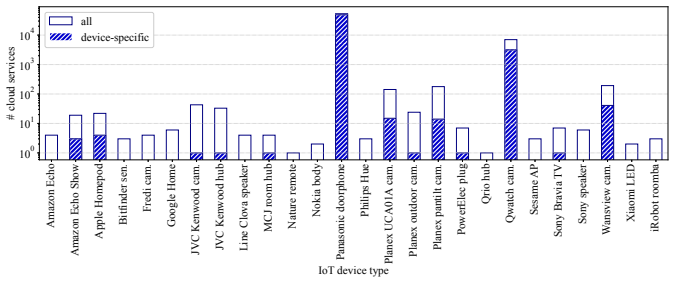



Fig. 9. Count of cloud services per IoT device type in our training dataset.

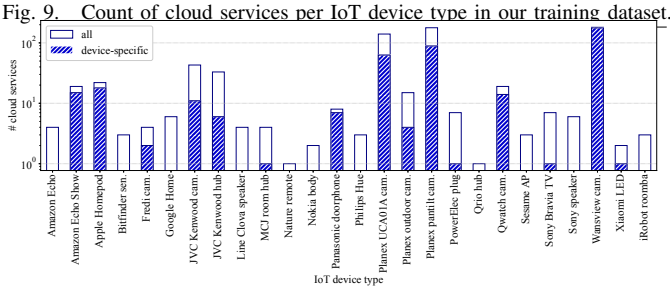



Fig. 10. Count of cloud services after filtering NAT traversing flows.

challenges: (a) They do not provide perfect accuracy. The best model would still result in some false positives [40], depending on the complexity of patterns across classes and the amount of training data available. For our problem, a false positive arises when our model detects an IoT device that is not present on the network. As explained in §III, false positive is detrimental for device inference at scale; (b) Activity behaviors of IoT devices are susceptible to firmware upgrade [32], which can significantly impact the stochastic features described in Table II. In that case, the stochastic model may require re-training with a new dataset to learn certain changes in benign behaviors of the upgraded device; and (c) Stochastic features can vary [8] depending on user activities and interactions with their devices at home. For example, considering a smart speaker, asking about the weather, or listening to a music track would result in different amounts of traffic generated on the network. Therefore, capturing a superset of all IoT behaviors at scale may not be practical.

On the other hand, IoT cloud services seem promising to overcome certain limitations of the stochastic model. Cloud services that certain IoT device types may share can be detected deterministically, reducing false positives. As cloud services are maintained on the servers (of manufacturers and/or application developers) and not on the device side, they are less likely to change by firmware upgrades or user activities. We denote a cloud service by a two-tuple information of `IP protocol/port number` (*e.g.,* `TCP/80`, or `UDP/53`) readily available in IPFIX records, that is not impacted by NAT operations. In fact, services that are exposed by IoT devices may also be helpful in detecting them; however, as they are overwritten by NAT routers and are not accessible post-NAT, hence beyond the scope of this paper.

### A. Fingerprint of IoT Cloud Services

Let us begin by considering two groups of IPFIX records, namely "outgoing" flows initiated by IoT devices inside the home network to cloud servers on the Internet and "incoming" flows initiated by remote endpoints (on the Internet) to household IoT devices. Cloud services, therefore, are captured from the source transport-layer port of incoming records and the destination transport-layer port of outgoing records.

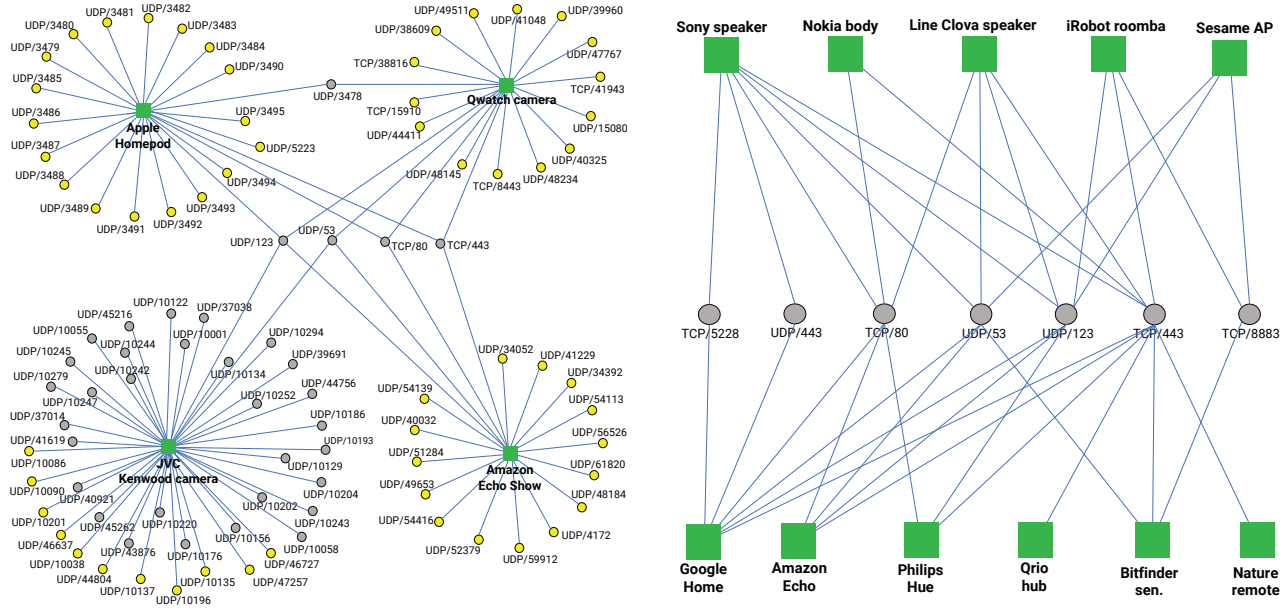
TABLE V
DISTRIBUTION OF CLOUD SERVICES ACROSS IOT DEVICE TYPES AFTER FILTERING NAT TRAVERSING FLOWS.

| | **Generic** | **Mixed** | **Specific** | **Total** |
|---|---|---|---|---|
| # services | 4 | 110 | 410 | 524 |
| # flows | 914,157 | 138,077 | 233,474 | 1,285,708 |

Fig. 9 shows the distribution of cloud services across various IoT device types in our training dataset (*i.e.,* January and first half of March). We find a total of 56,561 services under three groups: (1) ***generic*** services like `TCP/80` and `UDP/53` that are found in traffic records of more than half of our IoT devices; (2) ***mixed*** services that are shared across device types (only less than half of them) in our dataset – for example, `UDP/37038` is only shared between JVC Kenwood camera and hub; and (3) ***specific*** services that are only used by a single device type – service `UDP/41229` is only seen in flow records of Amazon Echo Show. Note that some device types, such as Fredi camera, Google Home, and Amazon Echo come with no specific service (only generic and/or mixed services). In contrast, device types like Qwatch camera and Panasonic doorphone display thousands of specific services. Observing such an undesirable behavior *i.e.,* very wide distribution of cloud services for some devices like Qwatch camera and Panasonic doorphone led us to further analyze their corresponding flow records.

A deeper investigation revealed that NAT traversing flows are the main cause for wide ranges of cloud services in the behavior of those IoT device types. Devices like smart cameras provide live video streams to their users. This feature requires the camera to be accessible behind NAT, which can be achieved either by having a relay server or directly traversing through NAT. In the former case, the server receives feed from the camera and passes it to users. In the latter case, user directly connects to the camera from the Internet, streaming the video feed – direct connections initiated from outside home networks may not be feasible for all implementations of NAT gateways [41]. If an IoT device behind NAT is accessible from outside, there will be communications flows destined to the device (*i.e.,* incoming flows), likely sourced from random transport-layer port numbers (cloud services). Therefore, we need to filter cloud services deduced from incoming flows and generate our deterministic model only based on cloud services obtained from outgoing flows (*i.e.,* initiated by IoT devices), avoiding random cloud services.

Filtering the incoming flows reduced the total number of cloud services to 5200. However, a device like Qwatch camera contributes to 4680 of cloud services. We further analyzed IPFIX records of Qwatch camera and noticed that this device uses Session Traversal Utilities for NAT (STUN) protocol [42]. STUN provides internal hosts with a set of tools to detect NAT type when communicating with remote endpoints. Upon completion of STUN tests, Qwatch camera performs NAT hairpinning (loopback) – two hosts behind the NAT (on the local network) communicate with each other via the external




(a) IoT device types with specific and shared services.

(b) IoT device types with only shared (generic and/or mixed) services.

Fig. 11. Fingerprint of IoT cloud services: (a) four representative device types with specific and shared services, and (b) 11 device types with only shared (generic and/or mixed) but no specific service. Bright nodes highlight specific services and gray nodes highlight shared services.

(public) IP address of the NAT gateway with port forwarding. We further exclude flows with public IP address of the home gateway to/from local IP addresses.

Fig. 10 shows the distribution of cloud services after removing incoming and hairpinning flows. It can be seen that the number of services (for certain device types) has significantly reduced by at least an order of magnitude (compared to Fig. 9) after filtering NAT traversing flows. For instance, the total count of cloud services for Qwatch camera and Panasonic doorphone has now become 19 and 8 services, respectively.

Table V summarizes the number of generic, mixed, and specific services along with their corresponding flow count. We can see a total of 524 cloud services across device types in our dataset where about 78% of them are specific to an IoT type. Four generic services, namely `TCP/80` (`HTTP`), `TCP/443` (`TLS`), `UDP/53` (`DNS`), and `UDP/123` (`NTP`) are shared between more than half of the device types. Note that generic services contribute to only 0.7% of discovered cloud services, but they are found in more than 70% of all flows analyzed.

We visualize in Fig. 11 two representative fingerprints of IoT cloud services: (a) Four device types (Apple Homepod, Qwatch camera, JVC Kenwood camera, and Amazon Echo Show) with over ten specific services; and (b) device types in our dataset that have no specific cloud service. It can be seen in Fig. 11(a) that even devices with a set of specific services (bright nodes) use a number of generic/mixed services (gray nodes) – no device type in our dataset comes with a pure set of specific services. Fig. 11(b) illustrates how device inference can be challenging with shared cloud services across different device types. For example, we can see that Sony speaker and Google Home have the exact same set of services. The same happens to be the case for Line Clova speaker and Amazon Echo. In another example, we observe that the fingerprint of Nokia body (*i.e.,* {`TCP/80, TCP/443`}) is a subset of five other fingerprints. We will develop a weighted inference technique (next subsection) that associates a device-specific weight to each cloud service proportional to its occurrence frequency to tackle overlapped fingerprints.

We note that the appearance and consistency of the fingerprint set over time can vary across device types in the training dataset. For example, the fingerprint set of Panasonic doorphone was fully captured during the first two weeks of our training dataset and remained consistent until the end of the training period (about a month and a half). In another example, Amazon Echo uses three of its four shared services (it has no specific service) every day it is active. On the other hand, some devices grow their fingerprint set over time. For instance, Wansview camera uses new specific services on four days in March that were not seen during January. While new services emerged, parts of the fingerprint remained consistent throughout the training period – for example, the camera communicated with its specific service `UDP/60722` every day it is active.

### B. Inference Techniques

Detecting devices based on their cloud services can be done in different ways. In this section, we discuss a couple of these possible methods.

**Specific Service**: This inference is made on a per-flow basis. We need to generate a map between individual cloud services and IoT device types from the training dataset for this method. For example, mappings `(TCP/1111)`$\rightarrow\{D_1\}$ and `(TCP/2222)`$\rightarrow\{D_2, D_3\}$ mean that the service `TCP/1111` exclusively belongs to the device type $D_1$ and the service `TCP/2222` is shared between device types $D_2$ and $D_3$. Then we apply the generated map to each of the individual testing flows. If a given flow matches a one-to-one mapping rule (corresponding to a specific cloud service), then the unique

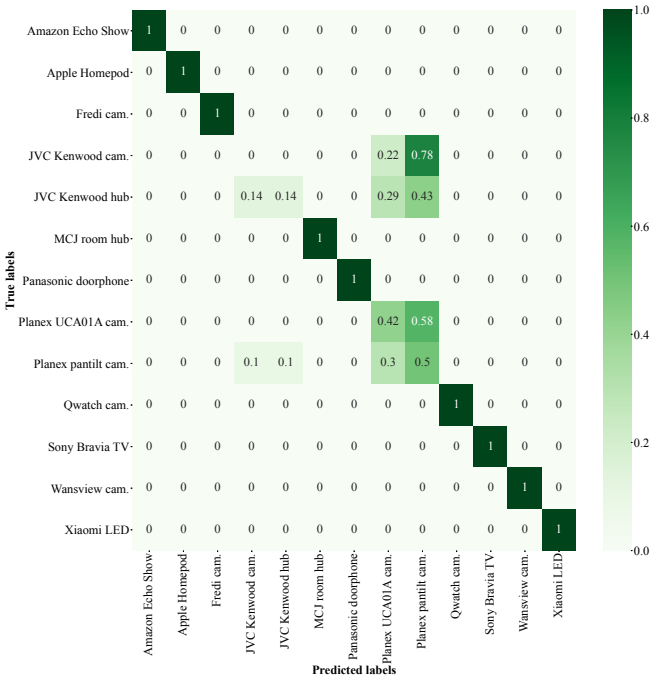



Fig. 12. Confusion matrix of our deterministic model with the specific service inference method.

type of the associated IoT device is inferred; otherwise if the cloud service is shared or unseen, no type is inferred.

**Fingerprint Set**: Inspired by [22], this inference is made based on a collection of flows observed during a configurable epoch (say, daily) per household. Therefore, deterministic fingerprints (each is a set of cloud services obtained from the training dataset) for individual device types, similar to what is shown in Fig. 11, are required as inputs to our inference. For testing during a given epoch, each cloud service (extracted from an IPFIX record) is checked against the fingerprint of all device types. If the service is found in a fingerprint set, we count a match for the corresponding device type (each service can be counted once per device type during an epoch). Note that a shared service (generic/mixed) will result in matches for more than one device type. At the end of the epoch, we compute the fraction of matched services for each device type. An IoT device is detected, if its computed fraction is above a configurable "fingerprint threshold".

The inference based on fingerprint sets can be employed in two ways: (a) "*unweighted*", whereby the membership of cloud services to various fingerprint sets is measured as a binary value so all services have the same contribution in computing the fraction of matched services, and (b) "*weighted*", whereby the service membership is measured by a weight (between 0 and 1), equal to the fraction of days that the service was seen in the training dataset over a total number of days on which the device was active. The final fingerprint match for each device is a weighted average of its observed services. That way, frequent services have more contribution in fingerprint matching. We consider the unweighted method as a baseline, representing deterministic models used in prior works like [15], [22].

We note that the specific service inference method would be faster in detecting devices as it responds to each individual flow record independently (stateless). The fingerprint set inference, on the other hand, is stateful and needs to analyze a collection of records observed during the epoch before a set of devices are detected. The specific service method can only detect those IoT devices that consume at least one specific service, which is not the case for 11 IoT device types in our dataset, as highlighted by purely white bars in Fig. 10.

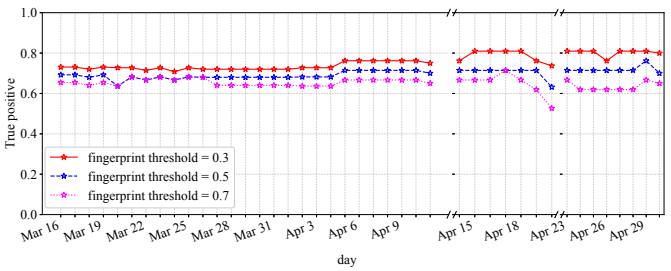



Fig. 13. Daily trace of true positives from the deterministic model based on unweighted fingerprint set method.

### C. Evaluating Efficacy of Deterministic Modeling

We use the same training dataset (*i.e.,* traces from the month of January and first half of March) we used for the stochastic model to generate the deterministic models (*i.e.,* map of specific services to device types and a fingerprint set for each device). Similarly, the traces from the second half of March and April are used for testing.

Before evaluation, let us briefly look at the composition of cloud services in the testing set. Overall, about 19% of testing flows have specific services that can be used to detect their corresponding IoT devices. A similar observation was made from the training dataset, reported by Table V. Among all class types, three cameras, including Fredi camera, Panasonic doorphone, and Wansview camera, significantly dominate by having more than 94% of their flows with services specific to their type. About 10% of the total flows have services shared between two or three device types. For example, more than three-quarters of flows for Qwatch camera are dedicated to the cloud service `UDP/3478` which is shared between Qwatch camera and Apple Homepod. That said, our specific service method (in what follows) will detect both the camera and homepod with a perfect accuracy based on their individual specific services. In another example, `TCP/8883` is shared between Bitfinder sensor (30% of flows), Sesame AP (98% of flows), and iRobot roomba (95% of flows). We will show (in what follows) how our fingerprint set method overcomes the challenge of device types without specific services. Lastly, about 0.02% of testing flows (*i.e.,* 268 IPFIX records) contain 150 new cloud services – not seen in the training dataset; 97 of them belong to Wansview camera. Such a dynamic in the fingerprint of Wansview camera was observed in the training dataset (§V-A), but it does not affect our inference – we will soon see that this camera type is perfectly detected by its consistent use of specific cloud services like `UDP/60722`.

**Evaluating the Specific Service Method**: As this method can only predict the device type of those flows that contain specific services, about 80% of testing flows remain unclassified. Fig. 12 shows the confusion matrix of this method. It

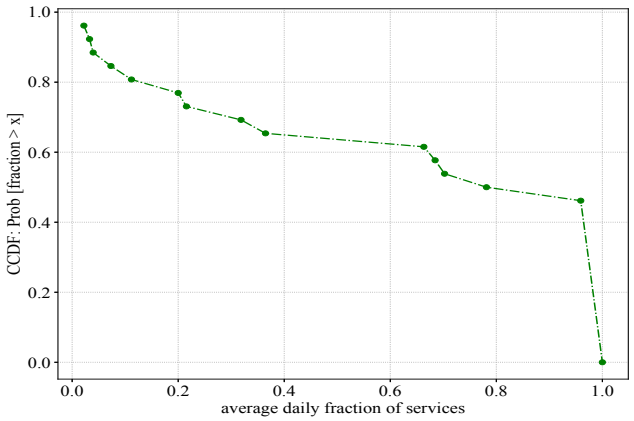



Fig. 14. CCDF of average daily faction of services per IoT device during the testing period.

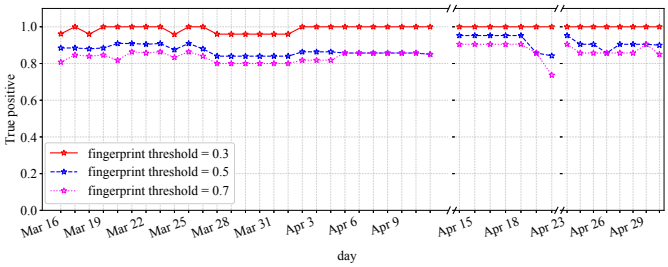



Fig. 15. Daily trace of true positives from the deterministic model based on weighted fingerprint set method.

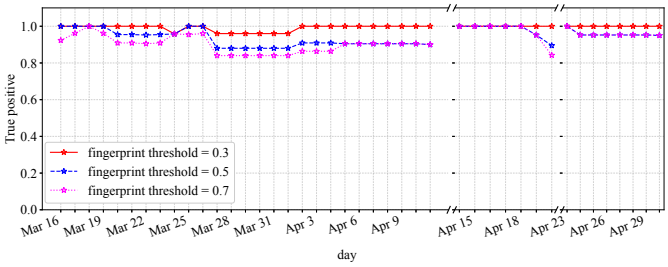



Fig. 16. Daily trace of true positives for the deterministic model based on the combination of specific service and weighted fingerprint set methods.

can be seen that the specific service method yields a perfect accuracy (100%) for nine device classes and an overall average of 77% across all classes of devices that have specific services. Note that the Planex outdoor camera and PowerElec plug do not use any of their specific services (found in the training dataset) during the testing phase; therefore, these two classes are excluded from the confusion matrix. Also, we observe that devices like the JVC Kenwood camera and hub started using cloud services that were initially specific to Planex cameras (UCA01A and pantilt), and hence noticeable misclassification rates for them. For instance, during the training phase, JVC Kenwood camera was found to use 11 specific services, but none of them were used during the testing phase. Instead, it communicated with 2 and 7 services specific to Planex UCA01A and pantilt cameras, respectively – this may be attributed to a firmware upgrade. This issue will be solved to some extent when we combine the two methods, namely specific service and fingerprint set.

**Evaluating the Fingerprint Set Method**: This method is able to cover some of the shortcomings of the previous method, particularly in predicting device types that only use shared services. For this method to work, we need to configure a fingerprint threshold on the fraction of observed cloud services to determine whether a corresponding device type is present on a given home network. Note that the fingerprint set method operates on a collection of records observed during an epoch (chosen daily, ensuring a quality inference). We, therefore, demonstrate the efficacy of this method across epochs during the testing phase.

Fig. 13 shows a daily trace of true positives (*i.e.,* ratio of correctly detected active devices to all active devices) obtained from our deterministic model (with the fingerprint set method in unweighted mode) during the testing period. For the fingerprint threshold, we tested three values of 0.3, 0.5, and 0.7 that respectively detect 75%, 70%, and 65% of active devices on average. Conservative operators may choose a higher value for the fingerprint threshold, providing them with a quality prediction at the cost of missing some devices types with weaker (shared) fingerprints. Setting the threshold value too low can be detrimental to the quality of inferencing, resulting in a high rate of false positives.

We found that the unweighted fingerprint set method can solve some of the shortcomings of the specific service method. For example, devices like the Bitfinder sensor, Sesame AP, and iRobot roomba which do not have any specific service, are correctly predicted on those days they are active. However, the detection rate at best reaches about 80% during the entire testing period. To better explain the root cause of such a performance, we measured the daily observed fraction of cloud services for each device – the fraction becomes equal to 1 if a subject device consumes all services from its fingerprint. We plot in Fig. 14 the CCDF of the average daily fraction of services (per IoT device) during the testing period. We found that the measure of daily fraction for five (out of 26) devices, including the JVC Kenwood camera, JVC Kenwood hub, Planex UCA01A camera, Planex pantilt camera, and Wansview camera never exceeded 0.2, and hence are consistently missed by our model in the unweighted mode. Interestingly, twelve devices receive an average fraction equal to 1 – they always get correctly predicted.

The unweighted mode turns out to be discriminating certain device types that have shared cloud services in their fingerprint since this method misses an important factor of the usage frequency. For example, a device type like Apple Homepod tends to use the generic service `TCP/80` every day, but the specific service `UDP/3488` only once or twice a week. Assigning a corresponding frequency weight to each fingerprint service can help our model better infer connected devices. Fig. 15 shows the daily trace of true positives obtained from the weighted method, given the same three fingerprint thresholds. It can be seen that giving more weight to frequent services can indeed improve the rate of true positives – on average, the detection rate is increased by 24%, 19%, and 20% respectively for threshold values of 0.3, 0.5, and 0.7. Specifically, devices like the JVC Kenwood camera and hub, with a relatively poor score ($< 0.2$) of the fingerprint match from the unweighted method, now receive about 0.64 and 0.79 match from the weighted method, and hence correctly predicted. Needless to say, JVC Kenwood camera consumed none of its specific cloud services,

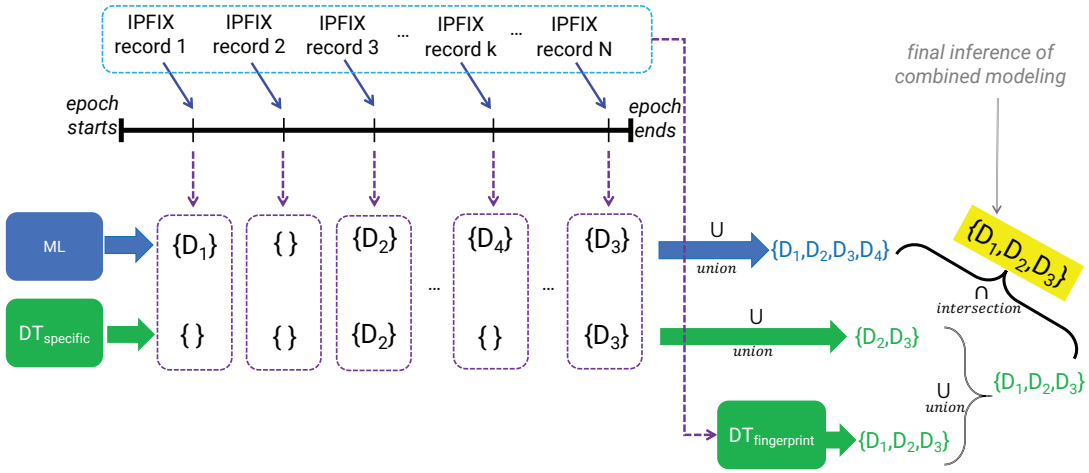



Fig. 17. High-level architecture of a combined inference by stochastic and deterministic models.

TABLE VI
POTENTIAL ADVANTAGES OF THE DETERMINISTIC MODEL, AUGMENTING THE STOCHASTIC MODEL.

| | Stochastic accepted | | Stochastic discarded | |
|---|---|---|---|---|
| | *correct* | *incorrect* | *correct* | *incorrect* |
| Total flows | 870K | 8K | 370K | 170K |
| specific-service flows [outgoing] | 223K | 3 | 10K | 645 |
| unseen-service flows [outgoing] | 155 | 1 | 74 | 38 |

and JVC Kenwood hub contacted one of its specific services during the testing period.

**Specific Service and Fingerprint Set Methods Combined**: We have seen that both methods display certain limitations for some device types or in some circumstances. The specific service method offers a narrow coverage by excluding shared cloud services. On the other hand, the (weighted) fingerprint set method can fall short in predicting a device, which does not consume a sufficient number of its expected cloud services during the inference epoch. To benefit from their respective differentiated advantages, we combine these two methods. Note that each operates differently (*i.e.,* the specific service method infers on a per IPFIX record basis, while the fingerprint set method infers from a collection of IPFIX records over an epoch). Therefore, we can combine their inferences at the end of each epoch by applying the union operation to the lists of predicted devices obtained from individual methods. That way, the chance of missing a connected IoT device will be minimized. We show in Fig. 16 the daily trace of true positives from our deterministic model when the two methods (specific service and weighted fingerprint set) are combined. It can be seen that the combined method further improves (compared to what was obtained in Fig. 15) the quality of inference, yielding an average true positive of 99%, 94%, and 92% for the three thresholds of 0.3, 0.5, and 0.7, respectively. The combined model helps detect devices like Planex pantilt camera which had a relatively low score (≈0.39 on average) of the fingerprint match with the weighted method only. Importantly, the combined method reduces the average rate of false positives to relatively low values (*i.e.,* 10%, 9% and 4%) for our chosen thresholds (0.3, 0.5, and 0.7, respectively).

### D. Combining Stochastic and Deterministic Models

We saw that the stochastic model gives a reasonable accuracy during the testing period (96% on average per class). However, it discards about 26% of the correctly classified (testing) flows due to a lack of confidence. Given its differentiated capabilities, the deterministic model has the potential to augment the stochastic model and improve the quality of inference. Table VI highlights, at a high level, the potential advantage of our deterministic model that can help the stochastic model. The first row summarizes how the stochastic model correctly/incorrectly classifies all flows (incoming and outgoing); some are accepted while others get discarded. The second row (highlighted in blue) shows a potential benefit of the deterministic model: (i) 223K (a quarter of) correctly classified and accepted flows can be verified as they use cloud services specific to certain types of IoT devices; (ii) 10K (≈3%) of correctly classified but discarded flows can be "revived". Note that (as discussed in §V-A) our deterministic model infers from outgoing flows (≈85% of all flows), those that originate from the home network. The third row (highlighted in yellow) shows some challenges that our deterministic model encountered with unseen services, but these types of flows were fairly negligible (less than 0.02%) in our dataset.

Fig. 17 shows the architecture of our combined inference from stochastic (*i.e.,* $ML$) and deterministic (*i.e.,* $DT$) models. We employ both specific service ($DT_{specific}$) and weighted fingerprint set ($DT_{fingerprint}$) methods for our deterministic inference. During a given epoch, the $ML$ and $DT_{specific}$ models work in parallel on a per-flow basis. Due to low confidence and/or an unseen/shared cloud service, some flows may result in no inference (*i.e.,* yielding an empty set) by either or both of the models. At the end of the epoch, the predicted set of devices is obtained by applying the union operation to their individual predictions during the epoch. Additionally, the third model ($DT_{fingerprint}$) gives its prediction on all flows observed during the epoch. Next, we obtain a superset of detected devices by computing the union of outputs from the two deterministic models. Lastly, we intersect the two predicted sets yielded by the stochastic and deterministic

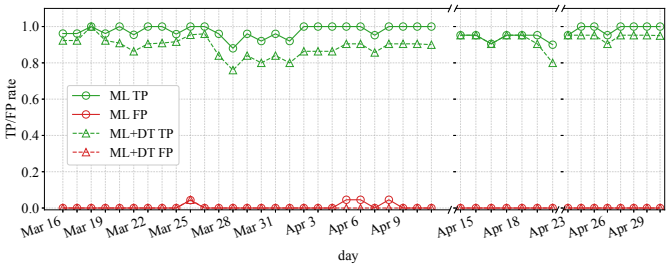



Fig. 18. Prediction performance of stochastic modeling versus combined stochastic and deterministic modeling.

TABLE VII
COMPARING PERFORMANCE OF STOCHASTIC AND DETERMINISTIC BASELINES WITH THAT OF THEIR COMBINATION.

| | **ML** (proxy of [3]) | **DT** (proxy of [22]) | **ML+DT** (this work) |
|---|---|---|---|
| TP | 97% | 92% | 90% [−7%] |
| FP | 0.4% | 4% | 0.1% [+75%] |

models, obtaining the final inference of connected IoT devices.

Fig. 18 illustrates how our combined inference impacts the performance of prediction by two metrics, namely rates of true positive (TP) and false positive (FP). TP indicates the correctly detected fraction of all devices that were active on the network, and FP indicates the incorrectly detected fraction of all active devices on the network. For this experiment, a comparison between two scenarios of stochastic inference ($ML$) versus the combined inference ($ML + DT$) is conducted. Also, we set the fingerprint threshold to 0.7 (a conservative setting) for our $DT_{fingerprint}$ model. We can see that the ML model is able to detect more devices on its own (with an average TP of 97%) compared to the combined scenario, given the conservative but confident nature of our deterministic model, a slightly lower TP (90% on average) is achieved. Instead, the conservative deterministic modeling helped us mitigate false positives generated by the ML model during three days (April 5, 6, and 8) of evaluation.

Table VII shows true and false positive rates for ML only (representing prior work [3]), DT only (representing prior work [22]), and ML+DT combined models (this work). ML outperforms DT in terms of true positives by about 5%. Also, the false positive rate of ML is ten times better than that of DT (0.4% compared to 4%). Higher false positives from the DT model are attributed to shared services explained in §V. Even though DT is less accurate than ML when the two models are combined (ML+DT), false positives significantly decrease to an attractive level of 0.1% – this measure is 75% better than that of ML only (highlighted in the green brackets). However, this improvement comes at the cost of losing about 7% of true positives from ML (highlighted in the red brackets).

The combined inference may miss some devices (one out of ten), but the chance of incorrectly detecting a device that is not present on the network is fairly low. As discussed throughout this paper, reducing false positives is more crucial as they can incur unwanted management costs (taking actions on a vulnerable device that does not exist in the user premise). Thus, losing 7% of true positives seems to be a reasonable trade-off for gaining 75% fewer false positives.

## VI. Conclusion

Residential networks continue to become richer and more vulnerable with the widespread adoption of consumer IoT devices. ISPs recognize the need to address concerns associated with IoT security by obtaining visibility into these connected devices and their behavior. This paper discussed how an ISP could leverage IPFIX telemetry to achieve this task post-NAT, at scale, without making changes to home networks. We analyzed about three million IPFIX records (released as open data) from 26 IoT devices. We extracted 28 flow-level stochastic features for training a machine learning model to detect device types from IPFIX records with an average accuracy of 96%. In addition to the stochastic model, we developed deterministic models based on cloud services captured from outgoing IPFIX records, yielding an average accuracy of 92%. We demonstrated that combining stochastic and deterministic models can help reduce false-positive incidents by 75% for an average cost of 7% in true positives.

## References


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

[2] S. Grover *et al.*, "Peeking behind the NAT: An Empirical Study of Home Networks," in *Proc. ACM IMC*, Barcelona, Spain, Oct 2013.

[3] Y. Meidan *et al.*, "A Novel Approach For Detecting Vulnerable IoT Devices Connected Behind a Home NAT," *Computers & Security*, vol. 97, pp. 1–23, Oct 2020.

[4] Statista, "Number of Smart Homes Forecast Worldwide from 2017 to 2025," 2020. [Online]. Available: https://bit.ly/3ulFflm

[5] Palo Alto Networks, "Unit 42 IoT Threat Report," 2020. [Online]. Available: https://start.paloaltonetworks.com/unit-42-iot-threat-report

[6] D. Kumar *et al.*, "All Things Considered: An Analysis of IoT Devices on Home Networks," in *Proc. USENIX Security*, Santa Clara, USA, Aug 2019.

[7] J. Anand *et al.*, "PARVP: Passively Assessing Risk of Vulnerable Passwords for HTTP Authentication in Networked Cameras," in *Proc. ACM DAI-SNAC*, Virtual Event, Germany, Dec 2021.

[8] A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE TMC*, vol. 18, no. 8, pp. 1745–1759, Aug 2019.

[9] T. Micro, "Mirai Botnet Exploit Weaponized to Attack IoT Devices via CVE-2020-5902," 2020. [Online]. Available: https://bit.ly/2RRLy1X

[10] H. Kumar *et al.*, "Enhancing Security Management at Software-Defined Exchange Points," *IEEE TNSM*, vol. 16, no. 4, pp. 1479–1492, Sep 2019.

[11] J. Blythe *et al.*, "What is Security Worth to Consumers? Investigating Willingness to Pay for Secure Internet of Things Devices," *Crime Science*, vol. 9, no. 1, pp. 1–9, Jan 2020.

[12] N. Nthala *et al.*, "Rethinking Home Network Security," in *Proc. EuroUSEC*, London, England, Apr 2018.

[13] D. Harkin *et al.*, "Consumer IoT and its under-regulation: Findings from an Australian Study," *Policy & Internet*, vol. 14, no. 1, pp. 96–113, 2022.

[14] H. Habibi Gharakheili *et al.*, "Cloud Assisted Home Networks," in *Proc. ACM CAN*, Incheon, Republic of Korea, Dec 2017.

[15] H. Guo *et al.*, "IoTSTEED: Bot-side Defense to IoT-based DDoS Attacks (Extended)," USC/Information Sciences Institute, Tech. Rep. ISI-TR-738, Jun. 2020. [Online]. Available: https://bit.ly/3ec9eGS

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

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

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

[19] A. Hamza *et al.*, "Combining MUD Policies with SDN for IoT Intrusion Detection," in *Proc. ACM IoT S&P*, Budapest, Hungary, Aug 2018.

[20] Cyber Edge, "Cyberthreat Defense Report," 2020. [Online]. Available: https://bit.ly/3xM1Idi

[21] A. Sivanathan *et al.*, "Can We Classify an IoT Device using TCP Port Scan?" in *Proc. IEEE ICIAS*, Colombo, Sri Lanka, Dec 2018.

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

[23] A. Hamza *et al.*, "Verifying and Monitoring IoTs Network Behavior using MUD Profiles," *IEEE TDSC*, vol. 19, no. 1, pp. 1–18, Jan-Feb 2022.

[24] A. Pashamokhtari *et al.*, "IoT IPFIX Records," 2021. [Online]. Available: https://iotanalytics.unsw.edu.au/iotipfixrecords

[25] A. Engelberg *et al.*, "Classification of Encrypted IoT Traffic Despite Padding and Shaping," in *arXiv preprint arXiv:2110.11188*, 2021.

[26] X. Ma *et al.*, "Inferring Hidden IoT Devices and User Interactions via Spatial-Temporal Traffic Fingerprinting," *IEEE/ACM ToN*, pp. 1–15, Sep 2021.

[27] W. Zhang *et al.*, "Optimizing Federated Learning in Distributed Industrial IoT: A Multi-Agent Approach," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3688–3703, 2021.

[28] R. Hofstede *et al.*, "Towards Real-time Intrusion Detection for NetFlow and IPFIX," in *Proc. CNSM*, Zurich, Switzerland, Oct 2013.

[29] J. J. Davis, "Machine Learning and Feature Engineering for Computer Network Security," Ph.D. dissertation, Queensland University of Technology, Brisbane, Australia, 2017. [Online]. Available: https://eprints.qut.edu.au/106914/1/Jonathan_Davis_Thesis.pdf

[30] B. Trammell *et al.*, "Bidirectional Flow Export Using IP Flow Information Export (IPFIX)," RFC 5103, Jan. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5103.txt

[31] D. d. Reis *et al.*, "One-class Quantification," in *Proc. ECML PKDD*, Dublin, Ireland, Sep 2018.

[32] A. Sivanathan *et al.*, "Detecting Behavioral Change of IoT Devices Using Clustering-Based Network Traffic Modeling," *IEEE IoTJ*, vol. 7, no. 8, pp. 7295–7309, Aug 2020.

[33] NetSA CERT, "YAF," 2006. [Online]. Available: https://tools.netsa.cert.org/yaf/yaf.html

[34] ——, "Super Mediator," 2012. [Online]. Available: https://tools.netsa.cert.org/super_mediator

[35] R. Moskowitz *et al.*, "Address Allocation for Private Internets," RFC 1918, Feb. 1996. [Online]. Available: https://rfc-editor.org/rfc/rfc1918.txt

[36] M. Cotton *et al.*, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry," RFC 6335, Aug. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6335.txt

[37] E. Lior *et al.*, "Manufacturer Usage Description Specification," RFC 8520, Mar. 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8520.txt

[38] A. Hamza *et al.*, "Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity," in *Proc. ACM SOSR*, San Jose, USA, Apr 2019.

[39] M. S. Mahdavinejad *et al.*, "Machine Learning for Internet of Things Data Analysis: a Survey," *DCN*, vol. 4, no. 3, pp. 161–175, Aug 2018.

[40] M. A. Al-Garadi *et al.*, "A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1646–1685, Apr 2020.

[41] M. Westerlund *et al.*, "Comparison of Different NAT Traversal Techniques for Media Controlled by the Real-Time Streaming Protocol (RTSP)," RFC 7604, Sep. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7604.txt

[42] P. Matthews *et al.*, "Session Traversal Utilities for NAT (STUN)," RFC 5389, Oct. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5389.txt


**Arman Pashamokhtari** received his B.Sc. degree of Computer Engineering from the Amirkabir University of Technology in Tehran, Iran in 2019. He is currently pursuing Ph.D. degree in the area of computer networks from the University of New South Wales (UNSW) in Sydney, Australia. His research interests include programmable networks, IoT network traffic analytics and applied machine learning.

**Norihiro Okui** received the B.E. and M.E. degrees of Computer Science and Engineering from Waseda University, Japan, in 2010 and 2012, respectively. He joined KDDI in 2012. He is currently a research engineer of the Cyber Security Lab. in KDDI Research, Inc. His research interest includes cyber security for IoT.

**Yutaka Miyake** received the B.E. and M.E. degrees of Electrical Engineering from Keio University, Japan, in 1988 and 1990, respectively. He joined KDD (now KDDI) in 1990, and has been engaged in the research on high-speed communication protocols and secure communication systems. He received the Ph.D. degree in engineering from the University of Electro-Communications, Japan, in 2009. He is currently a research manager of Cyber Security Laboratory and a director of Information System and Security Department at KDDI Research, Inc.

**Masataka Nakahara** received the B.Eng. degree of Electrical Engineering and the M. Informatics degree of Graduate School of Informatics from Kyoto University, Japan in 2014 and 2016, respectively. He joined KDDI in 2016, and joined KDDI Research, Inc. in 2019. His current research interest includes cyber security for IoT.

**Hassan Habibi Gharakheili** received his B.Sc. and M.Sc. degrees of Electrical Engineering from the Sharif University of Technology in Tehran, Iran in 2001 and 2004 respectively, and his Ph.D. in Electrical Engineering and Telecommunications from the University of New South Wales (UNSW) in Sydney, Australia in 2015. He is currently a Senior Lecturer at UNSW Sydney. His research interests include programmable networks, learning-based networked systems, and data analytics in computer systems.